SYNOPSYS®

DesignWare minPower Components

User Guide

F-2011.09
September 2011

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, CRITIC, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, the Synplicity Logo, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping Ssytem, HSIMplus, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, StarRC, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (℠)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

PCI Express is a trademark of PCI-SIG.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Preface

1.1 About This Manual

This manual explains the use of the DesignWare minPower Components, and is intended for users of Synopsys synthesis tools. DesignWare minPower Components are part of the overall DesignWare IP Library, but are licensed separately.

These minPower Components are technology-independent, micro-architecture-level collection of low power datapath generators and reusable intellectual property blocks that are tightly integrated into the Synopsys synthesis environment.

1.1.1 Web Resources

For additional information about the DesignWare IP Library, see

- Product documentation for DesignWare IP Library products on the Web:

  http://www.synopsys.com/dw/dwlibdocs.php

- You can download the latest complete DWBB and minPower Components release electronically through Synopsys' Electronic Software Transfer (EST).

  http://www.synopsys.com/dw/buildingblock_dl.php

- For up-to-date information about the latest DesignWare Library IP and verification models, visit the DesignWare home page:

  http://www.designware.com

- You can find general Synopsys Licensing (SCL) information on the Web at:

  http://www.synopsys.com/Support/Licensing

1.1.2 Manual Overview

This manual contains the following chapters and appendixes:

Preface (this document)		Describes the manual and lists the typographical conventions and symbols used in it; tells how to get technical assistance.
"Introduction"	page 13	Introduces the DesignWare Library IP and briefly describes the DesignWare minPower Components.

“Using minPower: Basic Setup”	page 19	Provides a basic look at how to use the DesignWare minPower Components.
“Using minPower: Advanced”	page 35	Provides an advanced look at how to use the DesignWare minPower Components.
“Using Licensed Implementations”	page 55	Describes how to use Licensed implementation of DesignWare minPower Components.
“Standard Synthetic Operators”	page 101	Provides a list of HDL operators that are mapped to synthetic operators in the Synopsys standard synthetic library standard.sldb.
“minPower Application Notes”	page 63	Provides helpful hints and tip on getting the most from DesignWare minPower use.

1.1.3 Typographical and Symbol Conventions

Table 1-1 lists the conventions that are used throughout this document.

Table 1-1 Documentation Conventions

Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). `% cd $LMC_HOME/hdl`
`Monospace`	System-generated text (prompts, messages, files, reports). `No Mismatches: 66 Vectors processed: 66 Possible"`
Italic or `Italic`	Variables for which you supply a specific value. As a command line example: `% setenv LMC_HOME prod_dir` In body text: In the previous example, *prod_dir* is the directory where your product must be installed.
| (Vertical rule)	Choice among alternatives, as in the following syntax example: `-effort_level low | medium | high`
[ ] (Square brackets)	Enclose optional parameters: `pin1 [pin2 ... pinN]` In this example, you must enter at least one pin name (*pin1*), but others are optional ([*pin2* … *pinN*]).
TopMenu > SubMenu	Pulldown menu paths, such as: File > Save As …

1.2 Getting Help

If you have a question about using Synopsys products, please consult product documentation that is installed on your network.

You can also contact the Synopsys Support Center in the following ways:

- To telephone or open a call to your local support center using this page:

  http://www.synopsys.com/Support/GlobalSupportCenters

- Send an e-mail message to support_center@synopsys.com.

1.2.1 Additional Information

For more information on the DesignWare IP Library, refer to the following:

http://www.designware.com
or email us at: designware@synopsys.com
☎ or call (877) 4-BEST-IP (423-7847)

1.3 Comments?

To report errors or make suggestions, please send e-mail to:

support_center@synopsys.com.

To report an error that occurs on a specific page, select the entire page (including headers and footers), and copy to the buffer. Then paste the buffer to the body of your e-mail message. This will provide us with information to identify the source of the problem.

1 minPower QuickStart

The DesignWare minPower Components is a collection of low power datapath generators and reusable intellectual property blocks that are tightly integrated into the Synopsys synthesis environment.

1.1 Where Do I Get DesignWare minPower Components?

DesignWare minPower Components are included in your Design Compiler library. You can also download the latest complete DesignWare minPower Components release (including DWBB) electronically through Synopsys' Electronic Software Transfer (EST).

http://www.synopsys.com/dw/buildingblock_dl.php

Refer to the *DesignWare minPower Components Release Notes* for how to download the latest release.

1.2 minPower License

- Check that you are pointing to the correct license. `list_license` should list "DesignWare-LP".
- Check the DC-Ultra version. The first official release version of minPower was 2009.06-SP1.

You can find general Synopsys Common Licensing (SCL) information on the Web at:

http://www.synopsys.com/Support/LI/Licensing

1.3 Design Qualification

For minPower design qualification information and qualification commands, see "Qualifying Designs for minPower" on page 87.

1.4 minPower Setup and Flow

Setup: add dw_minpower.sldb to `synthetic_library`, `link_library`:

```
set synthetic_library "dw_foundation.sldb dw_minpower.sldb"
set link_library [concat  $link_library $synthetic_library]
```

Flow:

- Annotate the design with switching activity from previous RTL simulation before synthesis:

  ```
  read_saif -input <rtl>.saif;
  ```

- Enable Datapath Gating

  ```
  set power_enable_datapath_gating true;
  ```

- Compile the design:

  ```
  compile_ultra
  ```

- Generate reports to determine minPower effectiveness:

  ```
  report_power;
  report_area -designware
  report_resource -hier
  report_datapath_gating
  analyze_dw_power
  ```

1.5 minPower Considerations

DesignWare minPower Components reduce power in datapath intensive designs. Some important things to consider when enabling minPower in your design flow are listed below.

- You need good datapath content in your design with efficient extraction into large datapath blocks.
- Large bit width operations such as 16-bit multipliers or greater will benefit more from minPower.
- For best QoR, hierarchical boundaries should be ungrouped. Use the retiming feature in DC-Ultra instead.
- Manual pipeline insertion should be avoided whenever possible.
- minPower is most effective in producing switching and glitch aware structures when switching annotation is provided (SAIF file).
- A good technology library with Multi-Vt cells, a good mix of drive strengths, and datapath cells will give best results with minPower.

For more minPower optimization techniques, see "minPower Application Notes" on page 63.

2 Introduction

The DesignWare minPower Components is a collection of low power datapath generators and reusable intellectual property blocks that are tightly integrated into the Synopsys synthesis environment. Using DesignWare minPower Components allows transparent generation of datapath structures for low power and high performance during synthesis. With a growing number of parts available, design reuse is enabled, and significant productivity gains are possible.

The minPower Components collection can generate “low power” implementations of many basic datapath functions, and include IP that implement more advanced arithmetic and floating-point functions.

2.1 Key Features of minPower Components

The following list describes the key features of DesignWare minPower Components:

- **Innovative low power architectures:** DesignWare minPower Components includes the ability to generate unique low power datapath structures that can suppress switching and glitch activities.
- **Power costing and switching-activity-aware optimization:** Datapath tree structure and operand encoding are optimized according actual design characteristics to achieve power savings that can't be realized with conventional techniques. This is accomplished by including power and switching activity in to the cost function along with other design constraints.
- **Built-in Datapath Gating logic:** An intelligent scheme is implemented to isolate the operands when the output of the datapath structure is not required. Because of the up-front consideration to the timing context of the design, isolation logic is inserted and optimized such that there is no negative impact to the overall design. Many of the minPower Building Block IP have built-in isolation logic that is strategically placed on paths with minimal area overhead and no timing impact.
- **Low Power Instantiated IP:** Low power versions of frequently used instantiated IP are available with DesignWare minPower Components. The power optimizations are a result of many features such as manually efficient coding to maximize clock gating, support for datapath gating, etc. Pipelined components are provided with patented data tracking technology to reduce register switching for better dynamic power performance.
- **Design reuse benefit, existing IP, and 3rd party IP:** minPower can benefit not only new designs but also existing IP, reused IP and 3rd party IP. The main requirement is that the design must be synthesized again from RTL.

2.2 What is DesignWare minPower Components?

DesignWare minPower Components is a unique IP product that includes the generation of low power datapath architectures and low power instantiated IP. DesignWare minPower Components uses innovative power reduction techniques at the datapath architecture level to suppress switching, glitches, and promote the use of low-leakage cells in datapath circuits. Conventional low power design techniques use power gating to turn off a portion of the chip. While this is an effective technique to reduce power of unused circuits, it still does not address the intrinsic power consumption of circuits that must remain powered on.

Another common technique used by Design Engineers is to reduce the frequency of these circuits if they can meet the performance target, but this technique typically increases logic, which introduces additional power elements and spurious transitions propagated throughout the circuits. DesignWare minPower Components are designed to reduce power for datapath circuits that tend to be a dominant consumer of power, even while other circuits in the design are inactive.

2.3 What is Datapath?

Datapath circuits are used to perform calculation on the data. In real world applications, we use these types of calculation to produce audio, still image or video for multimedia. The more advanced the multimedia, the more intensive the calculation. We also use arithmetic algorithms to compress, encode, transform and modulate data before transmission, which means the data will have to be decompressed, decoded, or demodulated at the receiving end. The higher the data rate, the more demanding the arithmetic calculations have to be.

In other words, datapath is the essential elements in multimedia, signal processing, and data processing functions. These datapath structures are often found in wireless transceivers, audio processors, image and video processors, digital modulation blocks, and other signal processing and data processing functions. Because of the increasing demand on better user experience in consumer applications and higher data rates, the datapath content of modern applications have to increase to deal with the higher amount of arithmetic operation needed on the chips.

Datapath is often the dominant circuit when other portions of the chips are turned off for power optimization. Usually, the main CPU and on chip memory draw significant power, but they can be shut off more frequently. Some circuits need to remain powered on because they perform some essential functions in the design. These usually involves calculation. Examples are: wireless standby mode, where the circuit monitoring for incoming signals needs to remain powered ON even when the rest of the design can be powered OFF, audio/video playback mode, data in transmission mode, etc. The fact that these datapath structures need to remain powered ON presents an opportunity to realize additional power savings with better up-front selection of datapath architecture for power.

2.4 DesignWare minPower Applications

The DesignWare minPower Components will benefit applications in:

- Mobile multimedia, where size, weight and battery life are critical.
- High performance networking
- High speed signal processing
- Processor designs which can also be improved by minPower technology

- Power Compiler™ customers can benefit from this technology as the power savings offered by DesignWare minPower is additive to the benefits with Power Compiler.

2.5 A Library Strategy for High-Level Design

DesignWare minPower Components enable a high degree of automation in design reuse, making it possible for the Synopsys synthesis tools to transparently perform various high-level, low power optimizations.

DesignWare minPower Components benefit from all the optimization techniques such as resource sharing, pin permutations and arithmetic optimizations that DesignWare Library currently offers. For information on these resources, please refer to: "A Library Strategy for High-Level Design" in Chapter 1 of the *DesignWare Building Block IP User Guide*.

Similar to the Synthetic library for DesignWare Library, minPower uses a separate library:

```
dw_minpower.sldb
```

2.6 How Do I Use DesignWare minPower Components?

The basic setup of DesignWare minPower is described in "Using minPower: Basic Setup" on page 19.

DesignWare minPower IP can be included in a design either through *operator inference* or *component instantiation*. In operator inference, synthetic operators are automatically inferred from the presence of particular operators in your HDL code. In component instantiation, your HDL code explicitly instantiates a synthetic module. Detailed procedures for inference and instantiation are given in "Using minPower: Basic Setup" on page 19.

2.6.1 Inference

Operator inference occurs when the synthesis tool encounters an HDL operator whose definition ties it to a synthetic operator. DesignWare minPower finds the required synthetic operator, inserts it into your design, and performs high-level optimizations on the resulting netlist. The following is a list of operators that can inferred directly from the user's HDL code.

adders, subtractors, multipliers, comparators, SOP, selectops and shifters

To characterize the implementations for comparison, the synthesis tool creates a pre-optimized model for each one. The timing and area characteristics of the models serve as the basis for implementation selection. To avoid duplicate work, the models are stored in a UNIX directory called the *synthetic library cache*. "Using minPower: Advanced" on page 35 includes a discussion of cache-management issues.

The implementation that best meets the optimization goals you have set for the entire circuit is selected. The chosen implementation is inserted, by default, as a level of hierarchy in your design, which is then mapped to the target technology and optimized.

It is possible for you to steer or even override the automated implementation-selection process. "Using minPower: Advanced" on page 35 details this advanced feature.

2.6.2 Instantiation

Synthetic modules are also explicitly instantiatable. This is sometimes necessary because inferring using the HDL Compiler family of tools is available only for operations that can be realized with combinational logic. Instantiation is possible for any synthetic module.

The following example shows one way to instantiate the parameterized synthetic module `DW01_add` in VHDL. The lines of code that refer to the adder module are shown in bold.

```
library IEEE,DWARE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW01_add_inst is
generic ( inst_width : NATURAL := 8 );
  port ( inst_A   : in std_logic_vector(inst_width-1 downto 0);
         inst_B   : in std_logic_vector(inst_width-1 downto 0);
         inst_CI  : in std_logic;
         SUM_inst : out std_logic_vector(inst_width-1 downto 0);
         CO_inst  : out std_logic );
end DW01_add_inst;

architecture inst of DW01_add_inst is
begin

  -- Instance of DW01_add
  U1 : DW01_add
  generic map ( width => inst_width )
  port map ( A => inst_A, B => inst_B, CI => inst_CI,
             SUM => SUM_inst, CO => CO_inst );
end inst;
```

The following example shows how to instantiate the same synthetic module in Verilog:

```
module DW01_add_inst( inst_A, inst_B, inst_CI, SUM_inst, CO_inst );
  parameter width = 8;
  input [width-1 : 0] inst_A;
  input [width-1 : 0] inst_B;
  input inst_CI;
  output [width-1 : 0] SUM_inst;
  output CO_inst;

  // Instance of DW01_add
  DW01_add #(width)
    U1 (.A(inst_A), .B(inst_B), .CI(inst_CI), .SUM(SUM_inst), .CO(CO_inst) );
endmodule
```

When the synthesis tool encounters the reference to `DW01_add`, it attempts to resolve the reference by looking for a module of that name in the available synthetic libraries.

When the synthesis tool finds the appropriate synthetic module, it looks for all the implementations that are associated with that module, and performs implementation selection.

2.7 Where Do I Get the DesignWare minPower Components?

DesignWare minPower Components are part of the DC synthesis image that installs with DC, although in order to use these structures and components, you must have a minPower (DesignWare-LP) license.

You can download the latest complete DC Library release, which includes DesignWare minPower Components, electronically through Synopsys Electronic Software Transfer (EST).

http://www.synopsys.com/dw/buildingblock_dl.php

Refer to the minPower Release Notes for detailed instructions on how to download the latest release.

You can find general Synopsys Common Licensing (SCL) information on the Web at:

http://www.synopsys.com/Support/LI/Licensing

3 Using minPower: Basic Setup

You include DesignWare minPower Components in your designs by inference or by instantiation. The use of DesignWare minPower Components enables you to take full advantage of high-level optimization, automatic implementation selection, and design re-use.

The primary tasks involved in using this IP include the following:

3.1 DesignWare minPower Components QuickStart

Setup: add dw_minpower.sldb to synthetic_library, link_library:

```
set synthetic_library "dw_foundation.sldb dw_minpower.sldb"
set link_library [concat  $link_library $synthetic_library]
```

Flow:

- Annotate design with switching activity from RTL simulation before synthesis (SAIF file):

  ```
  read_saif -input rtl.saif
  ```

- Compile:

  ```
  compile_ultra
  ```

3.2 Setting Up Your Environment

To make DesignWare minPower Components accessible to the Synopsys tools, you specify paths to the necessary synthetic library files and design library directories.

Synthetic operators and modules for many basic arithmetic and logic functions are defined in the Synopsys standard synthetic library, `standard.sldb`. These include IP that support built-in HDL functions. This library, and the associated design libraries, are set up for you as part of the Synopsys software installation.

3.2.1 Accessing Synthetic Libraries

To access modules in synthetic libraries other than `standard.sldb`, you must set two `dc_shell-t` variables: `synthetic_library` and `link_library`.

The `synthetic_library` variable is analogous to the `target_library` variable in technology libraries. Set `synthetic_library` as a list of `.sldb` files that you want to use in the `compile_ultra` command. When synthetic operators and modules are processed during a compile_ultra command, the operators, bindings, modules, and implementations of the designated library or libraries are used. Synthetic libraries are searched in the order that they are listed. If two modules in different libraries have the same name, the module in the first library listed is used.

As with target technology libraries, you must include synthetic libraries in your `link_library` variable.

Attention

When using the DesignWare minPower Components, you must include the `dw_minpower.sldb` synthetic library in your synthetic library list.

For example, to use the synthetic library `my_synlib.sldb` and the technology library `my_techlib.db`, set the following `dc_shell-t` variables:

```
dc_shell-t> set target_library [list my_techlib.db]
dc_shell-t> set synthetic_library [list dw_minpower.sldb]
dc_shell-t> set link_library [list dw_minpower.sldb]
```

Note You do not need to set your `synthetic_library` variable or your `link_library` variable to include `standard.sldb`. This library is included by default.

You cannot disable the standard synthetic library, but you can disable individual modules and implementations by using the `set_dont_use` command. To replace a particular implementation, disable it with `set_dont_use` and provide a substitute from another synthetic library (the `set_dont_use command` is explained in "Disabling Selected Synthetic Modules and Implementations" on page 40).

3.2.2 Accessing Design Libraries

To use a synthetic library, you must specify the location of the design library that contains the netlists and HDL files for the corresponding implementations.

Note Paths to Synopsys-supplied design libraries are established as part of the software installation procedure. You need to define design library paths explicitly only if you are working with design libraries that do not come from Synopsys.

You can define a design library in either of two ways: you can use the `define_design_lib` command during a `dc_shell-t` session, or you can set up a mapping from the library name to a UNIX path name in your design library file. The design library file, by default, is `.synopsys_sim.setup`.

3.2.2.1 Using the define_design_lib Command

The `dc_shell-t` command `define_design_lib` maps a library name to the design library directory. The syntax is

```
define_design_lib library_name -path directory
```

library_name is the name of the design library, and `-path` *directory* is the name of a UNIX directory.

For example, the synthetic library `dw_minpower.sldb` refers Design Compiler to implementations in the design library. The following command defines the path where these implementations are located:

```
dc_shell-t> define_design_lib dw_foundation
 -path [format "%s%s" $synopsys_root "/dw/dw_foundation/lib"]
```

`synopsys_root` is the Synopsys root directory. This command maps the name dw_foundation to the path `$SYNOPSYS/dw/dw_foundation/lib`.

You can implement `define_design_lib` commands in your `.synopsys_dc.setup` file.

3.2.2.2 Using the Design Library File

The other method of mapping design library names to UNIX directory paths is to use your design library file. This file contains mapping information in the following format:

```
library_name : directory_name
```

For example, the path for the `my_synthlib` design library is defined by

```
my_synthlib : $SYNOPSYS/dw/my_synthlib/lib
```

By default, the design library file is named `.synopsys_sim.setup`. See the *System Installation and Configuration Guide* for a description of this file.

You can specify a different file to be your design library file by setting the `design_library_file` variable in your `.synopsys_dc.setup` file.

3.2.3 Other Set-up Options

There are various other DesignWare-related set-up options you can use to configure your environment. Options are available for

- *Releasing HDL Compiler Licenses* – Implementation selection requires a Synopsys HDL Compiler license (either HDL Compiler for Verilog or VHDL Compiler). In previous releases, the HDL license was kept checked out through the whole process of compilation. A new variable, `hdl_keep_license`, was added to allow you to release the HDL Compiler license after the implementation-selection phase.

  When `hdl_keep_license` is `FALSE`, the HDL Compiler license is kept only during implementation selection. For compatibility, the default value of `hdl_keep_license` is `TRUE`. To economize license use, set `hdl_keep_license` to `FALSE` in your `.synopsys_dc.setup` file.

- *Setting Up the Cache* – The synthesis tools create component models during implementation selection. To avoid duplicating work, they cache these models in a UNIX directory called the synthetic library cache. For a description of cache setup options, see "Maintaining the Synthetic Library Cache" on page 45.

3.3 Displaying Information

The commands `report_synlib` and `report_design_lib` show you the contents of the synthetic libraries and design libraries that are currently accessible. The command `get_attribute` shows which subdesigns in a design hierarchy came from synthetic libraries.

3.3.1 Reporting the Contents of Synthetic Libraries

To determine the contents of a synthetic library, use the `report_synlib` command. The resulting report lists all of the operators and their pins; followed by all the modules with their pins, parameters, attributes, implementations and bindings.

The syntax of `report_synlib` is

```
report_synlib library_name [list_of_modules]
```

The arguments and command-line options are

library_name

Name of the library to report on.

list_of_modules

Restricts the report to the modules in the list.

Implementations are annotated to show the attributes specified for them, including legality and priority formulas and priority set IDs.

The library must either be loaded into `dc_shell-t`, or accessible through the `search_path`. The command `list -libraries` displays the names of libraries that are currently loaded.

See the `report_synlib` man page for further details.

The following example illustrates the use of the `report_synlib` command on the synthetic library `dw_minpower.sldb`.

```
dc_shell-t> report_synlib dw_minpower.sldb
****************************************
Report : library
Library: dw_minpower.sldb
Version: C-2009.06-SP2
Date   : Wed Oct 14 09:48:59 2009
****************************************

Library Type            : Synthetic
Tool Created            : C-2009.06-SP2
Date Created            : 08.27.09
Library Version         : C-2009.06:C-2009.06-DWBB_0909:ffc285b4

Synthetic Modules:

   Module
   -----------------------------------------------------------------------
   DW_lp_pipe_mgr   design_library: DW03
                    HDL parameter: stages = 2
                    HDL parameter: id_width = 2
                    Parameter: census_width = log2(stages+1)
```

3.3.2 Identifying the Contents of Design Libraries

To determine the packages, entities, and architectures defined in your design libraries, use the `report_design_lib` command. The syntax of this command is

```
report_design_lib [-libraries][-designs][-architectures]
                  [-packages][library_names]
```

The command-line options are:

- `libraries`

  Lists the library names and their respective UNIX directories, but does not list their contents.

- `designs`

  Lists the designs in libraries (Verilog modules, VHDL entities, or configurations).

- `architectures`

  Lists the designs in libraries and their architectures (which correspond to implementations in Synthetic Libraries).

- `packages`

  Lists the VHDL packages contained in libraries.

- *library_names*

  Is the list of libraries whose contents are to be displayed. If `library_names` is not used, all libraries are displayed by default.

The design library report also lists

- parameterized designs (designs that can perform functions on data of varying sizes)
- the most recently analyzed architecture of each design
- files without a source
- outdated files

If you execute the `report_design_lib` command without options, the command lists all contents of all available design libraries.

The following example shows the use of the `report_design_lib` command to list the mapping information and contents of one user's current design libraries, which are `WORK`, `SYNOPSYS`, `IEEE`, `DW01`, and `DW02`.

```
dc_shell-t> report_design_lib
****************************************
Report : hdl libraries
Version: 2009.06
Date   : Fri Jun 12 10:52:04 2009
****************************************
Contents of current design libraries
 DEFAULT (/usr/remote/designs/WORK)
 WORK (/usr/remote/designs/WORK)
 entity : p DWSL_addov
 architecture : DWSL_addov(cla)
 architecture :  m DWSL_addov(proprietary)
 architecture : DWSL_addov(rpl)
 DW01 (/synopsys/sparc/dw/dw01/lib)
 entity : DW01_ADD_AB
 architecture :  m DW01_ADD_AB(str)
 entity : DW01_ADD_AB1
 architecture :  m DW01_ADD_AB1(str)
```

```
 ...
 DW02 (/synopsys/sparc/dw/dw02/lib)
 entity : p DW02_booth
 architecture : DW02_booth(sim)
 architecture :  m DW02_booth(str)
 package : DW02_components
 ...
 IEEE (/synopsys/sparc/packages/IEEE/lib)
 package : GS_Types
 package : STD_LOGIC_SIGNED
 package : STD_LOGIC_UNSIGNED
 package : std_logic_1164
 ...
 SYNOPSYS (/synopsys/sparc/packages/synopsys/lib)
 package : ATTRIBUTES
 package : BV_ARITHMETIC
p --- This design has parameters.
m --- This architecture is the most recently analyzed.
```

3.3.3 Identifying Synthetic Objects in a Design

It can be useful to detect the presence of synthetic objects in designs. For example, you might want to understand the mapping that occurred from your HDL operator to a synthetic operator. Another example would be to see which implementation of a module Design Compiler chose, then determine whether to set the implementation manually.

The `get_attribute` command in `dc_shell-t` can report the presence of certain synthetic objects. Table 3-2 shows which objects `dc_shell-t` can detect and related attributes to arch for in a design.

Table 3-2 Synthetic Objects Detectable in Designs

Synthetic Object	Attribute to Search For
synthetic operators	is_synlib_operator
synthetic modules	is_synlib_module

For details of the attributes and command syntax, see the `get_attribute` man page.

The following example shows a query of the cell `U1` and reference `DW01_add_width5` for the attribute `is_synlib_module`.

```
dc_shell-t> get_attribute [list U1 DW01_add_width5] is_synlib_module
Performing get_attribute on cell 'U1'.
Performing get_attribute on reference 'DW01_add_width5'.
{"true", "true"}
```

3.4 Inferring IP with HDL Operators

One method of incorporating DesignWare minPower Components into your design is to use an HDL operator whose definition maps it to a synthetic operator. Inferred synthetic operators enter into the high-level optimizations performed by HDL Compiler. (See "Introduction" on page 13.)

This is the procedure:

1. Use the HDL operator in your design description.

2. Analyze and elaborate your HDL file.
3. Set constraints and compile (compile_ultra).

During compilation, Design Compiler automatically selects the appropriate synthetic module and implementation to perform the operation.

Note

To infer arithmetic operations in VHDL on non-numeric types, you need to declare the `std_logic_arith` package and use the Synopsys-defined data types `SIGNED` and `UNSIGNED` from that package.

Suppose you want to infer an adder in your design by using the HDL operator for addition. You can perform the following tasks to accomplish this.

3.4.1 Task 1: Inferring an Adder in Your Description

The following example is a VHDL design description that uses an addition operator (+) to infer an adder module.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity  DW01_add_oper is
  generic(wordlength: integer := 8);
  port(in1,in2 : in STD_LOGIC_VECTOR(wordlength-1 downto 0);
    sum : out STD_LOGIC_VECTOR(wordlength-1 downto 0));
end DW01_add_oper;

architecture oper of DW01_add_oper is
  signal in1_signed,in2_signed,
         sum_signed: SIGNED(wordlength-1 downto 0);
begin
  in1_signed <= SIGNED(in1);
  in2_signed <= SIGNED(in2);
  -- infer the "+" addition operator
  sum_signed <= in1_signed + in2_signed;
  sum <= STD_LOGIC_VECTOR(sum_signed);

end oper;
```

Design Compiler selects the appropriate module and implementation to use.

You have the option of specifying in advance which module and implementation will be selected. Instructions for manual control of implementation selection are given in "Using minPower: Advanced" on page 35.

The following example is a Verilog design description that uses an addition operator (+) to infer a synthetic module (an adder).

```
module DW01_add_oper(in1,in2,sum);
  parameter wordlength = 8;

  input [wordlength-1:0] in1,in2;
  output [wordlength-1:0] sum;

  assign  sum = in1 + in2;

endmodule
```

3.4.2 Task 2: Analyzing and Elaborating Your File

You use the `analyze` command to translate your HDL code into a form that the Synopsys tools can use. Each HDL design unit gives rise to one or more files in the target design library. You then use the `elaborate` command to link the analyzed design units into a single design database.

In the following example, the VHDL file `addition.vhdl` is analyzed and stored in the design library `LIB`. (To analyze the Verilog file `addition.v` you would use the `-format verilog` option instead of `-format vhdl`.) The `elaborate` command then builds the design.

The report_cell command, which lists the synthetic library cells used in a design, shows that the HDL addition operator (+) has been replaced by the appropriate synthetic operator.

```
dc_shell-t> analyze addition.vhdl -format vhdl -library LIB
/usr/remote/designs/addition.vhdl:
1
dc_shell-t> elaborate addition -lib LIB
Current design is now 'addition'
1
dc_shell-t> report_cell
****************************************
Report : cell
Design : addition
Version: 2009.06
Date   : Fri Jun 12 10:52:04 2009
****************************************
Attributes:
b - black box (unknown)
h - hierarchical
n - noncombinational
r - removable
s - synthetic operator
u - contains unmapped logic
Cell    Reference    Library    Area    Attributes
--------------------------------------------------------
plus    *ADD_TC_OP_8_8_8        0.00    s, u
--------------------------------------------------------
Total 1 cells                   0.00
1
```

Attention When working with unmapped synthetic operators (prior to compilation), do not use the extract command, or try to write your design out in VHDL or Verilog format. Design Compiler does not recognize synthetic operator names as valid HDL identifiers. You may not be able to simulate these HDL files or read them back into Design Compiler

3.4.3 Task 3: Setting Constraints and Compiling

In the following example, constraints are set and compilation is initiated.

```
dc_shell-t> set_output_delay 4 [all_outputs]
Performing set_output_delay on port 'sum[7]'.
...
1
dc_shell-t> compile_ultra
...
1
```

The next example shows reports generated after compilation.

The report_cell command shows that the synthetic operator has been replaced by the implementation of a synthetic module. addition is implemented with a single hierarchical cell with an area of 69 gate-equivalents.

The report_resources command displays a resource sharing report and an implementation report for addition. The Current Implementation section shows that Design Compiler selected the cla implementation of the synthetic module DW01_add to build this cell.

```
dc_shell-t> report_cell
****************************************
Report : cell
Design : DW_lp_piped_fp_add_inst
Version: C-2009.06-PRE_PROD
Date   : Sun May  3 19:50:44 2009
****************************************

Attributes:
   BO - reference allows boundary optimization
    b - black box (unknown)
   bo - allows boundary optimization
    h - hierarchical
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell               Reference       Library             Area  Attributes
--------------------------------------------------------------------------
U1                 DW_lp_piped_fp_add_inst_DW_lp_piped_fp_add_0
                                                   3694.680059
                                                          BO, bo, h, n
--------------------------------------------------------------------------------
Total 1 cells                                        3694.680059
1

dc_shell-t> report_resources
****************************************
Report : resources
Design : DW_lp_piped_fp_add_inst
Version: C-2009.06-PRE_PROD
Date   : Sun May  3 19:50:44 2009
****************************************

Resource Report for this hierarchy
============================================================================
| Cell     | Module              | Parameters         | Contained Operations  |
============================================================================
| U1       | DW_lp_piped_fp_add | sig_width=23       | U1                    |
|          |                     | exp_width=8        |                       |
|          |                     | ieee_compliance=0  |                       |
|          |                     | op_iso_mode=0      |                       |
|          |                     | id_width=8         |                       |
|          |                     | in_reg=0           |                       |
|          |                     | stages=4           |                       |
|          |                     | out_reg=0          |                       |
|          |                     | no_pm=1            |                       |
|          |                     | rst_mode=0         |                       |
============================================================================

Implementation Report
============================================================================
|                |                     | Current          | Set              |
| Cell           | Module              | Implementation   | Implementation   |
============================================================================
| U1             | DW_lp_piped_fp_add | rtl              |                  |
============================================================================
...
```

3.5 Instantiating IP

Another way to incorporate a DesignWare minPower Components in your design is to instantiate a synthetic module explicitly. This is the procedure:

1. Include a reference to the synthetic module in your description.
2. Analyze and elaborate your file.
3. Set constraints and compile (compile_ultra).

During compilation, Design Compiler automatically selects the appropriate implementation for the module based on the constraints you set.

You have the option of specifying in advance which implementation will be selected. Instructions for manual control of implementation selection are given in "Using minPower: Advanced" on page 35.

Suppose you want to instantiate an adder in your design. You can perform the following tasks to accomplish this.

3.5.1 Task 1: Including a Reference to an Adder in Your Description

The following example shows how to instantiate the parameterized synthetic module `DW01_add` in VHDL. The lines of code that refer to the adder module are shown in bold.

```
library IEEE,DWARE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW01_add_inst is
  generic(wordlength: integer := 8);
  port(in1, in2: in STD_LOGIC_VECTOR(wordlength-1 downto 0);
       ci      : in STD_LOGIC;
       sum     : out STD_LOGIC_VECTOR(wordlength-1 downto 0);
       cout    : out STD_LOGIC);
end DW01_add_inst;

architecture inst of DW01_add_inst is
begin
  -- instantiate DW01_add
  U1: DW01_add  generic map(width => wordlength)
    port map(A => in1, B => in2, CI => ci, SUM => sum, CO => cout);
end inst;
```

The following example shows how to instantiate the parameterized synthetic module `DW01_add` in Verilog. The lines of code that refer to the adder module are shown in bold.

```
module DW01_add_inst(in1, in2, cin, sum, cout);
  parameter wordlength = 8;
  input [wordlength-1:0] in1, in2;
  input cin;
  output [wordlength-1:0] sum;
  output cout;
```

```
  // Instantiate DW01_add
DW01_add #(wordlength)
  U1(.A(in1), .B(in2), .CI(cin), .SUM(sum), .CO(cout));

endmodule
```

Since a specific implementation is not instantiated in this file, implementation selection is performed automatically by Design Compiler.

3.5.2 Task 2: Analyzing and Elaborating Your File

In the following example, the VHDL file `adder.vhdl` is analyzed and stored in the design library `LIB`. The `elaborate` command then builds the adder. The `report_cell` command shows that the synthetic module `DW01_add` has been instantiated as a black box (see the cell attributes).

```
dc_shell-t> analyze adder.vhdl -format vhdl -library LIB
/usr/remote/designs/adder.vhdl:
1
dc_shell-t> elaborate adder -lib LIB
Current design is now 'adder'
1
dc_shell-t> report_cell

****************************************
Report : cell
Design : adder
Version: 2009.06
Date   : Fri Jun 12 10:52:04 2009
****************************************
Attributes:
  b - black box (unknown)
  h - hierarchical
  n - noncombinational
  r - removable
  S - synthetic module
  u - contains unmapped logic
Cell    Reference         Library    Area    Attributes
--------------------------------------------------------
U1      DW01_add_width8              0.00    b, S
--------------------------------------------------------
Total 1 cells                        0.00
1
```

3.5.3 Task 3: Setting Constraints and Compiling

In the following example, constraints are set (using the `set_output_delay` command) and compilation is initiated.

```
dc_shell-t> set_output_delay 4 [all_outputs]
Performing set_output_delay on port 'sum[7]'.
...
Performing set_output_delay on port 'sum[0]'.
Performing set_output_delay on port 'carry_out'.
1
dc_shell-t> compile_ultra
...
1
```

The following example shows reports generated after compilation. The `report_cell` command shows that the synthetic module `DW01_add` has been implemented with a single hierarchical cell with area 80.

The `report_resources` command displays a resource sharing report and an implementation report for `adder`. The `Current Implementation` section shows that Design Compiler selected the `cla` implementation of the synthetic module `DW01_add` to build this cell.

```
dc_shell-t> report_cell
****************************************
Report : cell
Design : adder
Version: 2009.06
Date   : Fri Jun 12 10:52:04 2009
****************************************
Attributes:
  b - black box (unknown)
  BO - reference allows boundary optimization
  h - hierarchical
  n - noncombinational
  r - removable
  u - contains unmapped logicCell

         Reference    Library    Area    Attributes
-------------------------------------------------------------
U1       adder_DW01_add_8_0      80.00   BO, h
-------------------------------------------------------------
Total 1 cells                    80.00
1

dc_shell-t> report_resources
****************************************
Report : resources
Design : adder
Version: 2009.06
Date   : Fri Jun 12 10:52:04 2009
****************************************
Resource Sharing Report for design adder
============================================================
|          |        |            | Contained | Contained |
| Resource | Module | Parameters | Resources | Operations|
============================================================
| r28      |DW01_add| width=8    |           | U1        |
============================================================
Implementation Report
============================================================
|           |         |Current        | Set               |
|Cell       |Module   |Implementation | Implementation    |
============================================================
```

```
|U1          |DW01_add |cla             |                  |
============================================================
1
```

3.5.4 Pre-Compiling Subblocks

Compiling large subblocks can be a lengthy process. Pre-elaborating and pre-compiling subblocks speeds up the compilation.

The first step is to elaborate a subblock, then save the results. The following example shows the command you use to save the results of elaborating the `DW_ecc` IP.

```
elaborate DW_ecc -arch str -lib DW04 -p "8,5,0"
write -hier -o DW_ecc_8_5_0.db
```

The next time you link a design that contains the DW_ecc, Design Compiler automatically reads in and links the file `DW_ecc_8_5_0.db` without re-elaborating it.

You can also save the results of a compilation. The following example shows the commands to save the compilation of the `DW_ecc`.

```
elaborate DW_ecc -arch str -lib DW04 -p "8,5,0"
/* set design constraints  */
compile_ultra
dont_touch   DW_ecc_8_5_0
write -hier -o DW_ecc_8_5_0.db
```

On subsequent compilations, the `DW_ecc` will not be re-optimized. If you want to re-optimize the design for new constraints, you can do either of the following:

- remove the `DW_ecc_8_5_0.db` file and start over, or
- re-compile the subblock again

The following example shows how to re-compile the subblock again.

```
remove_attribute DW_ecc_8_5_0 dont_touch
compile_ultra
dont_touch   DW_ecc_8_5_0
write -hier -o DW_ecc_8_5_0.db
```

3.5.5 Optional: Using the VHDL Components Package

In VHDL, before a synthetic module can be instantiated in the architecture body, it must be declared as a component in the design's architecture. You can avoid the necessity of providing individual declarations for every synthetic module by declaring a library component package instead.

Every synthetic library from Synopsys includes a package that declares all designs in that library. The name of this package is `libraryname_components`, where `libraryname` is the name of the design library. If you declare this package with the `library` and `use` statements at the beginning of your VHDL description, you do not need to explicitly declare synthetic modules before you instantiate them.

For example, to declare all designs in the `DW01` library, include the following statements at the beginning of your code:

```
library DW01;
use DW01.DW01_components.all;
```

The following example demonstrates the use of this package (the required statements are shown in bold).

```
library IEEE, DW01;
use IEEE.std_logic_1164.all;
use DW01.DW01_components.all;

entity DW01_add_inst is
  generic(wordlength: integer := 8);
  port(in1, in2: in STD_LOGIC_VECTOR(wordlength-1 downto 0);
    ci: in STD_LOGIC;
    sum: out STD_LOGIC_VECTOR(wordlength-1 downto 0);
    cout: out STD_LOGIC);
end DW01_add_inst;

architecture inst of DW01_add_inst is
begin
  -- instantiate DW01_add
  U1: DW01_add
    generic map(width => wordlength)
    port map(A => in1, B => in2, CI => ci, SUM => sum, CO => cout);
end inst;
```

3.5.6 Viewing DesignWare minPower Components in Design Analyzer

When you display a schematic in Design Analyzer, DesignWare minPower Components are drawn on a separate layer called `designware_layer`. You can use the `set_layer` command in Design Compiler or the `View -> Style` menu in Design Analyzer to change the appearance of specific layers. You can modify the color, line width, and plot line width, and specify whether the layer will be visible or invisible.

3.6 Summary of Procedures for Using DesignWare minPower Components

1. To access synthetic libraries other than `standard.sldb`, you must include those libraries in your `synthetic_library` and `link_library` variables. For minpower, you need to add the library:

   ```
   dw_minpower.sldb
   ```

2. To access design libraries (other than those associated with standard.sldb), you must specify paths to the library directories using either the `dc_shell-t` command `define_design_lib`, or the mapping mechanism in the design library file `.synopsys_sim.setup`.
3. To list the operators, modules, and implementations available in a synthetic library, use the `report_synlib` command.
4. To list the packages, entities, and architectures available in a design library, use the `report_design_lib` command.
5. To infer DesignWare minPower Components:
   a. include the appropriate HDL operator in your design description,
   b. analyze and elaborate your design description (analyze and elaborate commands), and
   c. set constraints and compile (compile_ultra command with minPower).
6. To instantiate a synthetic module:
   a. include a reference to the module in your design description,

b. analyze and elaborate, and

c. set constraints and compile (compile_ultra).

7. When using VHDL, you can replace multiple component declarations with `library` and `use` statements that refer to the library components package. For example, to make all IP from the DW01 library available without having to declare each one, include the following statements in your VHDL design description:

```
library DW01;
use DW01.DW01_components.all;
```

4 Using minPower: Advanced

The Synopsys synthesis tools match HDL operators with synthetic operators and select implementations based on your constraints. During the process of implementation selection, the tools create timing models of implementations and cache them in a UNIX directory called the *synthetic library cache*. By default, all these processes are handled automatically.

To meet your specific requirements, you can use manual controls to guide or to override many of the automated processes associated with DesignWare minPower Components.

Advanced use of DesignWare minPower Components include:

4.1 Power-saving Methods of minPower Components

4.1.1 Introduction

The DesignWare minPower Components populate a library that implements techniques to produce power saving circuits in high-performance designs. The main features that can be enabled to minimize power consumption when using the Low Power Library components are as follows:

- Maximize clock gate insertion structures within components (page 35)
- Pipeline management of pipelined components (page 36)
- Datapath Gating for components with enable control (page 36)

4.1.2 Power Savings via Clock Gate Insertion

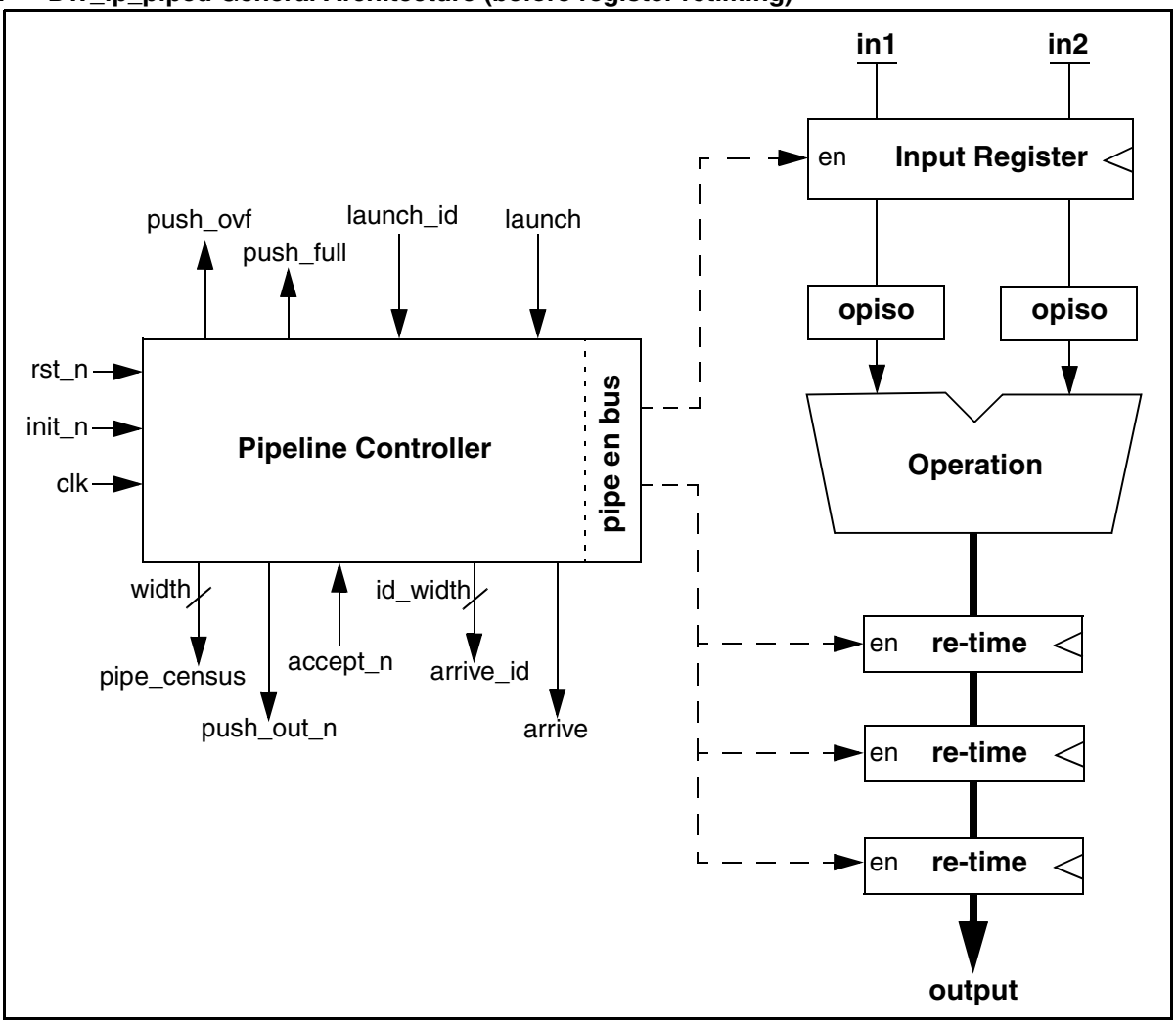
All low-power sequential and pipelined DesignWare components (DW_*_pipe and DW_lp_piped_*) are coded in such a way to allow clock gate insertion by DC synthesis. To enable this feature for DesignWare components, use the `-gate_clock` option to DC-Ultra and DC-Expert.

4.1.3 Pipeline Control Provides Power Savings (and enables Clock Gating)

In general, DW_lp_piped_* components contain a pipeline control block than runs in parallel to the pipeline register levels that monitors the activity through the pipeline (see Figure 4-1 block diagram for pre-register retiming depiction). In cases where there is inactivity on a particular register level of the pipeline, the pipeline control disables those levels to promote power savings. Furthermore, if using the Synopsys DC-Ultra or DC-Expert tools, the presence of the pipeline control and its wiring to the pipeline register levels provides an opportunity for increased power reduction in the form of clock gating (by setting the `-gate_clock` option).

As Figure 4-1 shows, many of the pipeline register banks are stacked back-to-back. This is the organization of these registers before register re-timing is applied by Design Compiler. The number of register levels is configurable depending on the performance required by the DesignWare component of interest.

Figure 4-1 DW_lp_piped General Architecture (before register retiming)



4.1.4 Datapath Gating on Sequential Components

For DesignWare components that have the Datapath Gating feature, the `op_iso_mode` parameter is provided. Datapath Gating can be applied in the form of 'and' or 'or' gates and the style is fixed for all

operands and bits of the operands once selected by the designer. Datapath Gating circuits, when inserted, are placed immediately after the input ports of the component. Datapath Gating is only allowed the opportunity to get inserted when there contains no input registers on the operands at the component boundary. Figure 4-1 shows a block diagram concept of how the operands are selected.

The application of Datapath Gating can be made in a global sense at synthesis compile time or locally overridden on a module-by-module basis via specific parameter value settings of the component. As mentioned previously, *op_iso_mode* is the parameter provided with each DesignWare component with Datapath Gating capability. The following table is the definition of the *op_iso_mode* component parameter.

Table 4-3 op_iso_mode Parameter Definition

Parameter	Valid Values	Description
op_iso_mode	0 to 4 Default: 0	Datapath Gating Selection During Design Compiler synthesis control global or local implementation selection of Datapath Gating. 0 = apply Design Compiler variable “DW_lp_op_iso_mode” 1 = 'none' (overrides “DW_lp_op_iso_mode” setting) 2 = 'and' (overrides “DW_lp_op_iso_mode” setting) 3 = 'or' (overrides “DW_lp_op_iso_mode” setting) 4 = preferred gating style: ‘and’ or ‘or’ depending on component

If the DesignWare component is instantiated with *op_iso_mode* set to '0', then during synthesis compile, the Design Compiler variable DW_lp_op_iso_mode will dictate which style of Datapath Gating is inserted, if any. If the DesignWare component is instantiated with *op_iso_mode* set to something other than '0', then the style of Datapath Gating applied is fixed; `none`, `and`, or `or` isolation. This mechanism allows the designer to selectively force Datapath Gating on a module-by-module basis before synthesis compile (by setting *op_iso_mode* to non-zero) and/or experiment with different Datapath Gating methods, in a global sense by setting *op_iso_mode* to '0'on the instance, and re-running synthesis without changing source code but with different DW_lp_op_iso_mode settings.

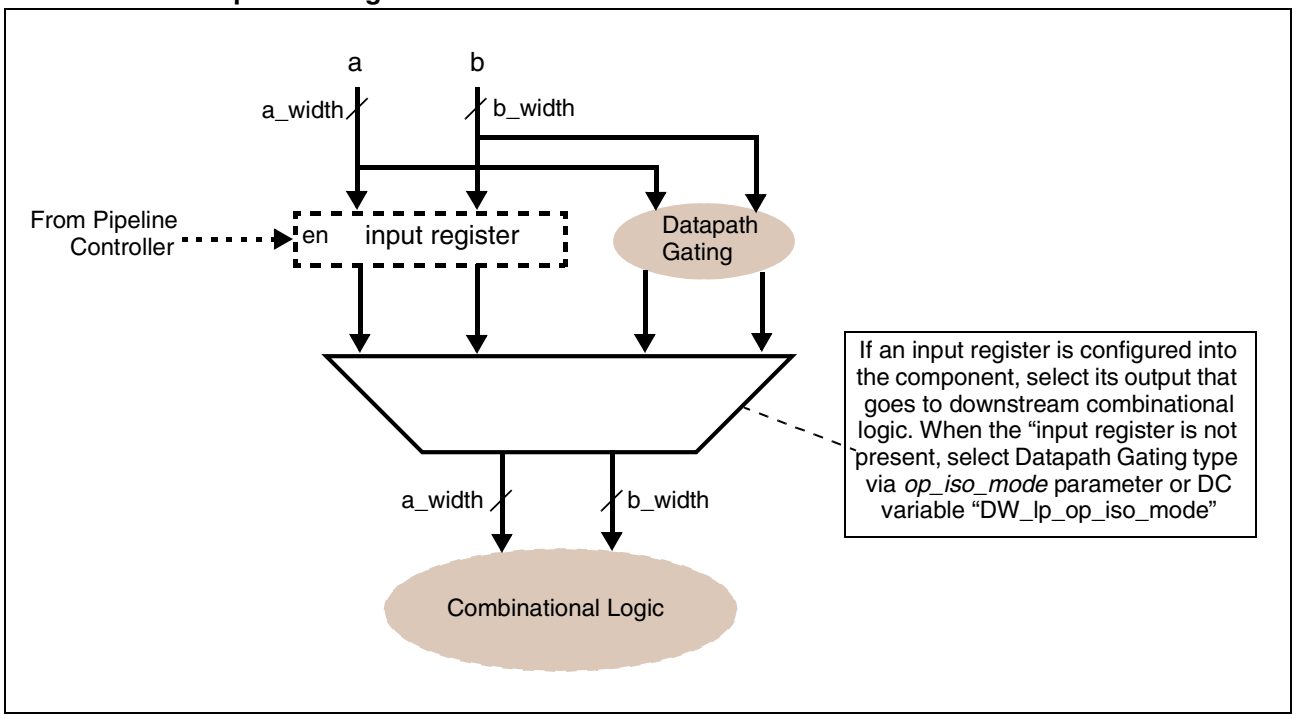
As an example, the following can be used to set up the synthesis scripts if the desire is to insert and gate-type Datapath Gating to DesignWare components configured with *op_iso_mode* as '0':

```
set DW_lp_op_iso_mode 2  # tcl syntax
```

Any DesignWare components instantiated with *op_iso_mode* set as non-zero will not be impacted by the setting made by the synthesis command and the local override will determine the method of Datapath Gating inserted.

Note: if the *op_iso_mode* parameter is set to '0' in a component and DW_lp_op_iso_mode is either not set during synthesis or set to '0', then no Datapath Gating will be inserted for the component.

Figure 4-2 When Datapath Gating Can Be Inserted



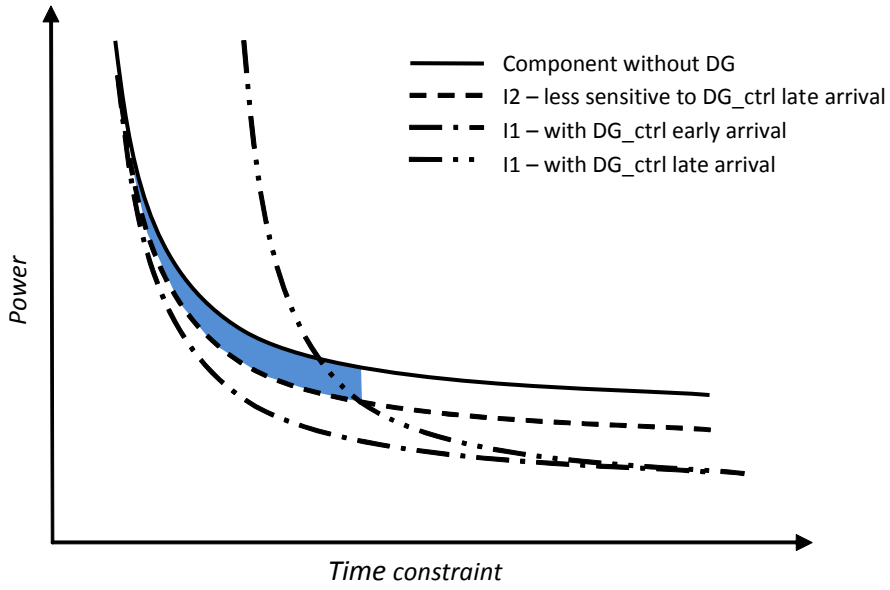
4.1.5 Datapath Gating on Combinational Components

Combinational building blocks may be gated to save dynamic power. The control pin for Datapath Gating in these blocks is called `DG_ctrl`. The component has the same behavior as its non-DG counterpart when `DG_ctrl=1`. When gated, the component has an arbitrary output value. The amount of power savings is highly dependent on the component's utilization and the distribution of the enable control signal. Higher power savings can be achieved when `DG_ctrl=0` for longer periods.

The reduction of dynamic power is obtained by clamping internal signals to a constant value. This operation may be done at different logic levels, depending on the implementations available for each component. Several of the combinational building blocks with Datapath Gating will have more than one implementation with different levels of gating. An implementation that is available for all the components in this category is the input gating (let's call it generically as I1). A second implementation (I2), when available, takes advantage of gating logic in the middle of the component's critical path.

When the `DG_ctrl` input arrives at the same time as all the other inputs, I1 is able to save more power than I2, for the same activity on the input ports. However, when `DG_ctrl` has a slightly late arrival, I1 will have increased power for the same overall delay constraint, while I2 will be less sensitive to this condition and consume less power than I1. A late arrival in `DG_ctrl` for implementation I1 is equivalent to a reduction in the required component's critical path, and therefore, more compile runtime and more area are used to reach the same target delay, with increased power consumption. Also, I1 will most likely not reach the same target delays as I2 when `DG_ctrl` is late. Figure 4-3 shows an example of a typical power versus time constraint curves for I1 and I2 when `DG_ctrl` has a late arrival or not. The power savings when using I2 for late arrival of `DG_ctrl` is shown as a shaded area in Figure 4-3.

Figure 4-3 Power vs. Time Constraint Curves of Typical DG Implementations



4.2 Controlling Module and Implementation Selection

During compilation, the synthesis tools automatically select the best implementation for each IP that occurs in your design. When you include DesignWare minPower Components by operator inference, the tools can choose among all the implementations of all the synthetic modules bound to the operator. When you instantiate a synthetic module, the tools can choose among the implementations of that module. This process of automatic *implementation selection* occurs (by default) each time you compile your design.

You have the capability, however, to control this process in various ways. You can do the following:

- Override the automatic selection and explicitly determine which synthetic module and implementation are used.
- Replace unmapped synthetic operators in your design with generic logic. See "Replacing Unmapped Synthetic Operators" on page 40.
- Prevent Design Compiler from selecting specified implementations. See "Disabling Selected Synthetic Modules and Implementations" on page 40.
- Prioritize the implementations of a given module. See "Prioritizing Implementations" on page 40.
- Control when implementation selection takes place. See "Controlling Incremental Implementation Selection" on page 41.

There are several reasons to override the automatic selection of modules and implementations as follows:

- You already have the specific implementation of a module in mind.
- You want a specific layout implementation for a hard macro (on the basis of area, aspect ratio, pinout, and so on).
- You want to select the lowest-power implementation of a module to keep overall system power consumption as low as possible.

You can also manually control resource sharing. For information on manual resource sharing, refer to Chapter 7 of the *HDL Compiler for Verilog Reference Manual* or Chapter 9 of the *VHDL Compiler Reference Manual*.

4.2.1 Replacing Unmapped Synthetic Operators

You can replace all the synthetic operators in your design by using the `replace_synthetic` command. `replace_synthetic` performs simple area-based implementation selection and resource sharing on your design. All synthetic operators are mapped to generic logic representations of selected modules. The syntax of the command is

```
replace_synthetic [-ungroup]
```

The `-ungroup` option ungroups all implementations into their containing designs.

Several high-level optimizations during compilation (including timing-driven resource sharing) depend on synthetic operators. Using `replace_synthetic` before compile disables these optimizations.

4.2.2 Disabling Selected Synthetic Modules and Implementations

You can disable individual synthetic modules and implementations with the `dont_use` command. For example, to disable the module `DW01_addsub` in `standard.sldb`, use the command:

```
set_dont_use standard.sldb/DW01_addsub
```

To disable a particular implementation of a module (`rpl`, in the example below), use an extra field:

```
set_dont_use standard.sldb/DW01_addsub/rpl
```

The `report_synlib` command lists the modules and implementations in a library, and shows which ones have a `dont_use` attribute.

4.2.3 Prioritizing Implementations

You can use *priorities* to express your preferences among the available implementations of a given module. A priority is an integer between 0 and 10. An implementation with no priority assigned to it has priority of 5 by default.

For a given module, only the highest-priority implementation (or implementations, in case of a tie) is considered for implementation selection.

Priorities distinguish only among implementations of the same module. The priorities of implementations of different modules are unrelated and hence cannot be used to establish preferences.

Implementation priorities can be coded (as functions of module parameters) into the library definition of a part. The `dc_shell-t` command `set_impl_priority` lets you override the priorities specified in a synthetic library without modifying the library itself.

The syntax of set_impl_priority is:

```
set_impl_priority [-priority priority] [-set_id id] implementation_list
```

The arguments and command-line options are:

-priority *priority*

Specifies the implementation priority. priority is a quoted integer between 0 and 10 (inclusive).

-set_id *id*

Non-negative integer that puts the implementation into a given set. Priority comparisons are made only between implementations in the same set. The default value is 0.

implementation_*list*

Names the implementation(s) whose priority you want to set.

For example, the following command specifies the priority for my_impl is "3" and that it belongs to set 2.

```
dc_shell-t> set_impl_priority my_lib/my_mod/my_impl "3" -set_id 2
```

You cannot use the set_impl_priority command in shell scripts embedded in HDL descriptions.

You can disable priority testing by setting the dc_shell-t variable hlo_ignore_priorities to "true".

When you begin to use a new library, or when you have questions about how implementation priorities are affecting your synthesis results, follow this procedure:

1. Issue the report_synlib command to generate a report that shows implementation priorities.
2. Use the set_impl_priority command, if necessary, to change priorities.

4.3 Controlling Incremental Implementation Selection

Implementation selection of synthetic modules occurs on every compile. However, Resource-sharing decisions remain fixed after the first compile.

Implementation selection will NOT be performed if you choose "map_effort low", unless you control the process manually by using the set_implementation command.

Implementation selection is controlled by the dc_shell-t variable, compile_implementation_selection, which defaults to "true". When the variable is set to "false", implementation selection is disabled and implementation decisions will not be changed from the current choice, unless you select an implementation manually by using the set_implementation command.

4.3.1 Effect on set_implementation

The set_implementation command interacts with incremental implementation selection in two ways:

- The values you set remain effective for all compiles, not just the first.

```
set_implementation impl_name cell_list
```

- The command syntax stays the same, but "." is now a valid `impl_name`:

  If you specify "." as the `impl_name` for a synthetic module that was mapped during a previous compile, the implementation for that module will not change in subsequent compiles. For example, the following command keeps the implementations of cells `U1` and `U2` fixed during subsequent compiles:

  ```
  set_implementation . { U1 U2 }
  ```

  To allow the implementation to be changed again, use the `remove_attribute` command:

  ```
  remove_attribute cell_list "implementation"
  ```

  For the example given above, the appropriate command is

  ```
  remove_attribute { U1 U2 } "implementation"
  ```

4.3.2 Effect on report_resources

The `report_resources` command generates two reports. The first report shows the results of resource sharing, but not implementation selection. The second report shows the implementations that are still in your design (ungrouped implementations are not listed). It tells you the current implementation for each instance, as well as any implementation selected by `set_implementation`. The resource sharing report does not change after the first compile; the implementation report reflects the results of incremental implementation selection.

The following example shows both reports.

```
prompt> report_resources -hierarchy

****************************************
Report : resources
Design : trunc3
Version: A-2007.12-SP1
Date   : Fri Jan  4 02:33:50 2008
****************************************

Resource Report for this hierarchy in file ./designs/trunc3.v
===============================================================================
| Cell           | Module         | Parameters | Contained Operations       |
===============================================================================
| add_x_9_0      | DW01_add       | width=6    | add_9                      |
| DP_OP_6_296    | DP_OP_6_296    |            |                            |
===============================================================================

Datapath Report for DP_OP_6_296
===========================================================================
| Cell                | Contained Operations                            |
===========================================================================
| DP_OP_6_296         | sub_8 add_8                                     |
===========================================================================
|       |      | Data     |       |                                       |
| Var   | Type | Class    | Width | Expression                            |
===========================================================================
| I1    | PI   | Unsigned | 5     |                                       |
| I2    | PI   | Unsigned | 5     |                                       |
| I3    | PI   | Unsigned | 5     |                                       |
| O1    | PO   | Signed   | 7     | I1 - I2 + I3                          |
===========================================================================

Implementation Report
===============================================================================
|                  |                  | Current          | Set              |
| Cell             | Module           | Implementation   | Implementation   |
===============================================================================
| DP_OP_6_296      | DP_OP_6_296      | str (power)      |                  |
| add_x_9_0        | DW01_add         | rpl (area,speed) |                  |
===============================================================================

No multiplexors to report

****************************************
Design : trunc3_DW01_add_0
****************************************

No resource sharing information to report.
No implementations to report
No multiplexors to report

****************************************
Design : trunc3_DP_OP_6_296_0
****************************************

No resource sharing information to report.
No implementations to report
No multiplexors to report
```

Note that the `Set Implementation` field may differ from the `Current Implementation` field. This happens when you have issued the `set_implementation` command, but have not yet compiled. Before the first compile, the report shows only `Set Implementation` information, since nothing else has been decided.

4.4 Controlling Hierarchy

After the `compile` command is completed, implementations are, by default, instantiated as a level of hierarchy in your design. For example, each adder is represented by an instance of an adder design.

In some cases, you can ungroup implementations, collapsing their contents into the containing design. Small arithmetic parts and all multiplexers supplied by Synopsys are automatically ungrouped so that they can be optimized with surrounding logic. Additionally, you can force modules to be ungrouped by using the `set_ungroup` command before you run `compile`.

4.5 Removing Unconnected Ports

Implementations of DesignWare minPower Components often have unused ports. For example, a particular implementation may not use all of the features of the IP, or the implementation may not use the entire bit-width of the IP parameterized input or output buses. An unused output port is an output that is not connected on the outside of the design. An unused input port is an input port that is not connected on the outside or is connected to a constant.

When a design is compiled with boundary optimization enabled, the logic that feeds an unused output port (or reads an unused input port) is optimized out of the design, leaving the port unconnected on the inside of the design. Unconnected ports, in turn, can cause problems for downstream tools (for example, layout tools or simulation tools).

To remove unconnected ports from the design, use the `remove_unconnected_ports` command. The syntax is:

```
remove_unconnected_ports [-blast_buses] cell_list
```

The `remove_unconnected_ports` command deletes the unconnected ports from the cells in `cell_list`. The command also deletes unconnected ports from the cell's reference and subdesigns. Because different cells with the same reference may be connected differently, you must `uniquify` the design before you execute `remove_unconnected_ports`.

By default, for bused ports, `remove_unconnected_ports` only deletes a bus when all members of the bus are unconnected. Otherwise, all members of the bus (connected and unconnected) are left intact.

If you specify the `-blast_buses` option, `remove_unconnected_ports` performs the following steps when it finds an unconnected port that is part of a bus:

1. Deletes the bus.
2. Deletes the unconnected ports.
3. Creates individual buses for the remaining connected ports to replace the deleted bus.

The following example removes all unconnected ports in the current design:

```
dc_shell-t> remove_unconnected_ports [find -hierarchy cell "*"]
```

The following example deletes buses that contain unconnected ports in the current design, deletes the unconnected ports, then creates individual buses for the remaining connected ports:

```
dc_shell-t> remove_unconnected_ports -blast_buses [find -hierarchy cell "*"]
```

The `remove_unconnected_ports` command does not affect cells that have the `dont_touch attribute` set.

4.6 Maintaining the Synthetic Library Cache

During implementation selection, Design Compiler compares the timing and area characteristics of different implementations. To perform the comparison, Design Compiler creates an optimized timing model for every implementation it considers.

To avoid repeating work, Design Compiler stores the models it creates in a UNIX directory called the *synthetic library cache*. When Design Compiler considers an implementation whose timing model is already in the cache, it does not have to create that model again.

This section describes the structure of the cache, tells you about the commands and variables available to you for controlling the cache mechanism, and provides tips for using the cache effectively.

4.6.1 Structure of the Cache

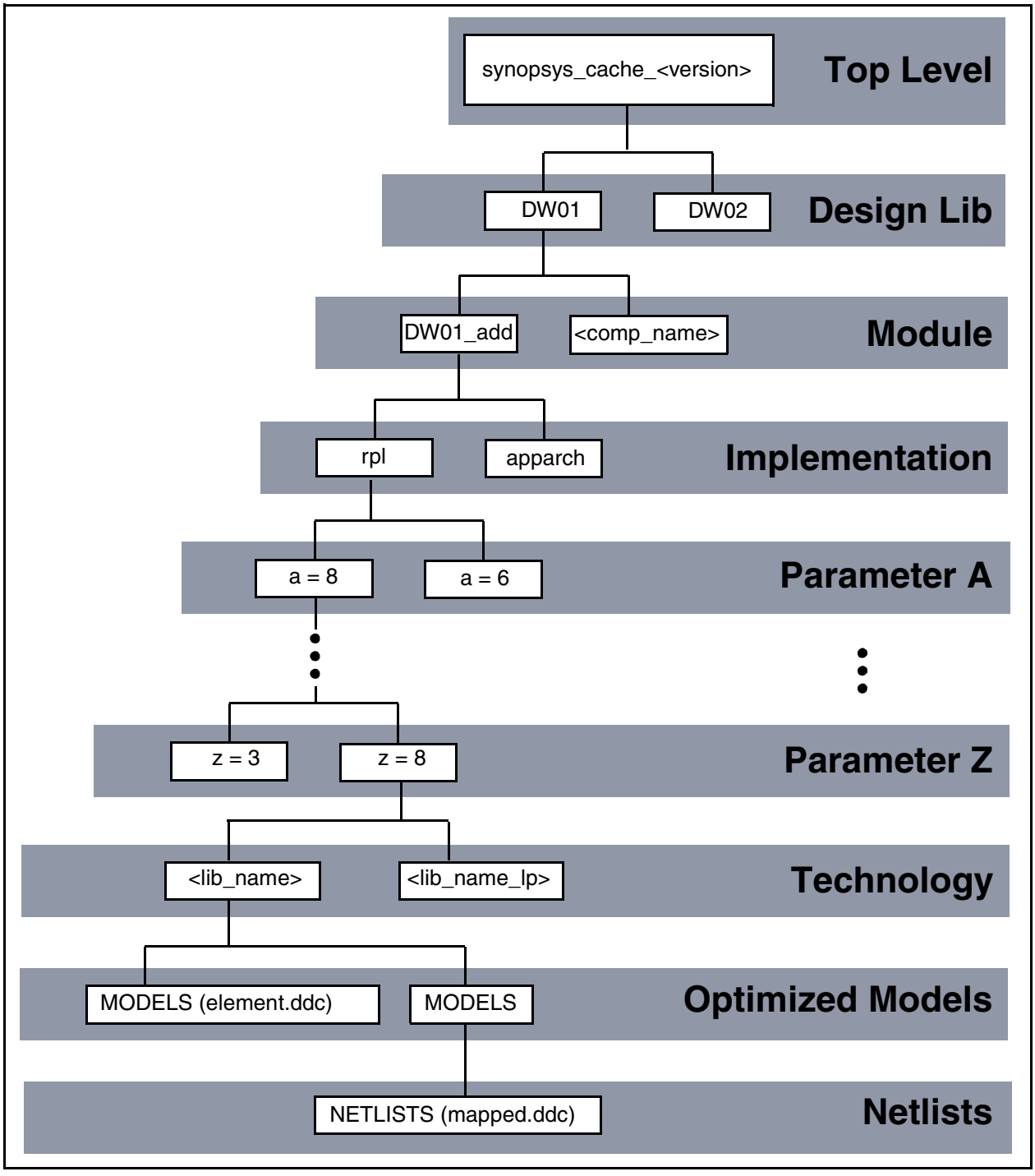
By default, the cache is created under your home directory. (For alternatives, see "Cache Variables" on page 51.) The cache is arranged as a hierarchical UNIX directory structure. `synopsys_cache_vX.X` is the top of the cache directory structure, where `vX.X` is the version number of Design Compiler.

Cache entries are indexed by design library name (DW01, for example), module, implementation, technology library, and parameters (from parameter 1 to parameter n). Figure 4-4 illustrates the cache directory hierarchy.

There are two types of elements stored in the cache: unoptimized netlists and optimized models. The cache directory structure separates netlists from models.

- Hierarchy for netlists – `design library, module, implementation, technology library, parameters, NETLIST, element.db.`
- Hierarchy for models – `design library, module, implementation, technology library, parameters, MODELS, wire load, operating conditions, element.db`

Figure 4-4 Directory Hierarchy of the Synthetic Library Cache



At the bottom of the directory tree structure, the models for the optimized implementations are stored as `.db` files.

4.6.2 Controlling the Cache

Three dc_shell-t commands give you control over the synthetic library cache:

- create_cache – Creates customized models and puts them in the cache, so you do not have long delays on your first compile.
- report_cache – Reports on the contents of the cache.
- remove_cache – Removes unwanted files from the cache.

These commands, and several variables related to cache control, are discussed in the following sections.

4.6.2.1 Creating the Cache Prior to Compilation

The create_cache command creates a model for each implementation that matches the conditions you specify on the command line. It uses the technology library specified by the target_library variable. The models are put in the directory specified by the cache_write variable. (See "Cache Variables" on page 51.)

With create_cache, you can create customized instances of synthetic parts in the cache without inferring or instantiating them through HDL descriptions. In this way, a design team leader can create a set of customized synthetic parts required for the current project, and place them in a (common) cache directory.

The syntax of create_cache is:

```
create_cache [-implementation list]
[-parameters parameter_list]
[-operating_condition string]
[-wire_load simple_list] [-report] -module list
```

The command-line options are:

```
-implementation list
```

Tells create_cache which implementations to use for each module. If you do not use this option, create_cache creates models for all available implementations. list is a space-separated list of strings. These strings can contain the wildcard character (*).

```
-parameters parameter_list
```

Specifies the parameter values create_cache will use when creating cache entries.

parameter_list is a space-separated list of quoted parameter specifications; a parameter specification is a comma-separated list of parameter settings. For example, {"N=8,M=6" "N=3"} is a valid parameter_list consisting of two parameter specifications.

Parameter specifications can include ranges of parameter values. For example, {"N = [8;17)"} means that N is greater than or equal to 8 and less than 17. See the create_cache man page for full details. The number and the name of the parameters you provide must match those of the given module(s); otherwise, you get an error message. The error message tells you the parameters you have to specify for each module.

```
-operating_condition string
```

Tells create_cache what operating conditions to assume. The operating condition you choose must be defined in the technology library specified by the target_library variable. If you do not use the -operating_condition option, create_cache assumes the default operating condition for that library.

```
-wire_load list
```

Tells `create_cache` what wire-load model to use. The wire-load model you choose must be defined in the technology library specified by the `target_library` variable. If you do not use the `-wire_load` option, `create_cache` uses the appropriate default wire-load model specified by that library.

`-report`

Generates and displays a very brief timing report for each created model.

`-module list`

Names the modules you want to put in the cache.

Some example command lines are

```
dc_shell-t> create_cache -mod DW01_add -p [list {width =4} -rep -o WCCOM
dc_shell-t> create_cache -mod *_mult -implementation [list wall csa] -par[list
{A_width =16, B_width =16} {A_width = (4;8], B_width = 6”}]
-oper WCCOM -wir [list “10x10” “90x90”]
```

Using wildcards in the `-module` and `-implementation` options can sometimes cause too many parts to be created; this can increase compile time. The parts are created, however, only if all the listed modules have the same set of parameters.

4.6.3 Reporting Cache Contents

The `report_cache` command reports on the cache directories listed in the `cache_read` and `cache_write` variables. The report lists the cache entries that match the conditions you specify on the command line.

The syntax of `report_cache` is:

```
report_cache [-design_lib list] [-module list]
[-implementation list] [-parameters parameter_list]
[-tech_lib list] [-wire_load list]
[-operating_conditions list] [-directory dir_list]
[-smaller size | -larger size]
[-accessed_since days | -accessed_beyond days]
[-netlist_only | -model_only]
[-sort_largest | -sort_oldest | -sort_cache_key]
```

The command-line options are:

`-design_lib list`

Restricts the report to the design libraries in the list. The default value for `design_lib` is the list of all currently defined design libraries. list is a space-separated list of strings. Strings in a list can contain the wildcard character (`*`).

`-module list`

Restricts the report to the modules in the list.

`-implementation list`

Tells `report_cache` which implementations to report on for each module. If you do not use this option, `report_cache` lists all available implementations.

`-parameters parameter_list`

Limits the report to implementations whose parameter values match the specified ranges.

parameter_list is a space-separated list of quoted parameter specifications; a parameter specification is a comma-separated list of parameter settings. For example, {"N=8,M=6" "N=3"} is a valid parameter_list consisting of two parameter specifications.

Parameter specifications can include ranges of parameter values. For example, {"N = [8;17)"} means that N must be greater than or equal to 8 and less than 17. See the report_cache man page for full details.

The braces { } are important: -parameters "n=8, m=6" is parsed as -parameters {"n=8" "m=6"}, not as -parameters {"n=8,m=6"}. To match a cache entry with two parameters, n and m, you need the braces.

The special parameter setting {"no_parameters"} matches entries with no parameters. For example, the specification {"N=8, no_parameters"} matches entries with no parameters or with N equal to 8.

-tech_lib list

Restricts the report to the technology libraries in the list. The default value for tech_lib is the value of the target_library variable.

-wire_load list

Restricts the report to the wire loads in the list. The wire-load model you choose must be defined in the technology library specified by the target_library variable. If you do not use the -wire_load option, create_cache assumes the default wire-load model for that library.

-operating_conditions list

Restricts the report to the operating conditions in the list.

-directory dir_list

Tells report_cache to search the directories on your list, instead of those specified in the read_cache and write_cache variables. dir_list is a space-separated list of strings. Wildcards are not allowed.

-smaller (-larger) size

Reports only files smaller (larger) than the specified value. size is an integer that gives the threshold size in bytes.

-accessed_since (-accessed_beyond) days

Reports only files whose last access occurred less than (more than) the specified number of days ago. days is a floating-point number.

-netlist_only (-model_only)

Reports only netlists (models).

-sort_largest (-sort_oldest, -sort_cache_key)

The default output format lists cache entries clustered by directory and design library. The options -sort_largest, -sort_oldest, and -sort_cache_key cause the report to be sorted by size, last access time, and cache-key value, respectively.

For a cache entry to be reported, it must be accepted by every input option. An entry is accepted by an input option if it matches one of the items in the input option's list. For example, the following command lists the ripple adder and the Wallace-tree multiplier.

```
command report-cache -module {DW01_add DW02_mult} -implementation {wall rpl}
```

Some combinations of the input options do not exist (ripple multipliers and Wallace-tree adders), but because each option acts as an independent filter, no difficulties arise.

A cache entry can meet a `-parameter` specification in two ways:

1. The names and values of the `parameter_list` match the cache entry parameters exactly.
2. All the cache entry parameters are matched in the `parameter_list` but the `parameter_list` contains additional parameters that do not apply to the entry.

Otherwise, the entry does not meet the `-parameter` specification.

The default output format lists cache elements clustered by directory, technology library, wire load, and operating conditions. Example 4-1 shows the default output resulting from the command

```
dc_shell-t> report_cache -larger 5000
```

4.6.4 Removing Items from the Cache

The syntax of the `remove_cache` command is identical to that of `report_cache`:

```
remove_cache [-design_lib list] [-module list]
[-implementation list] [-parameters parameter_list]
[-tech_lib list] [-wire_load list]
[-operating_conditions list] [-directory dir_list]
[-smaller size | -larger size]
[-accessed_since days | -accessed_beyond days]
[-netlist_only | -model_only]
[-sort_largest | -sort_oldest | -sort_cache_key]
```

`remove_cache` looks through the cache directories in the same way `report_cache` does, removing those entries that match your criteria. It also removes any empty directories it encounters.

Note The `remove_cache` command removes files from both the `cache_read` and the `cache_write` directories.

Example 4-1 Cache Report

```
****************************************
Report : cache
Version: 1998.02
Date : Wed Aug 31 14:17:03 1997
****************************************

================================================================================
| DESIGN     | MODULE       | IMPLEMENT-    | PARAMETERS | ACCESS |  SIZE |
| LIBRARY    |              | ATION         |            |   days | bytes |
================================================================================
Cache Directory Root: '/usr/remote/'
Technology Library: 'and_or'
Wire Load: no_wire_load
Operating Conditions: no_operating_conditions
```

```
| DW01       | DW01_ADD_ABC | str         |                | 0.0 |  5203 |
| DW01       | DW01_absval  | cla         | width=3        | 0.0 | 10064 |
| DW01       | DW01_add     | cla         | width=5        | 0.0 | 17376 |
| DW01       | DW01_inc     | cla         | width=3        | 0.0 |  9136 |
| DW01       | DW01_inc     | cla         | width=6        | 0.0 | 10912 |
| DW02       | DW02_mult    | csa         | A_width=3      | 0.0 | 29440 |
|            |              |             | B_width=3      |     |       |

================================================================================
Cache Directory Root: '/usr/remote/'
Technology Library: 'and_or'
Wire Load: '-default-'
Operating Conditions: '-default-'
| DW01       | DW01_ADD_ABC | str         |                | 0.0 |  6162 |
| DW01       | DW01_GP_SUM  | str         |                | 0.0 |  5302 |
| DW01       | DW01_MUX     | str         |                | 0.0 |  5134 |
| DW01       | DW01_XOR2    | str         |                | 0.0 |  5020 |
| DW01       | DW01_absval  | cla         | width=3        | 0.0 |  9768 |
| DW01       | DW01_add     | cla         | width=5        | 0.0 | 20982 |
| DW01       | DW01_inc     | cla         | width=3        | 0.0 | 10469 |
| DW01       | DW01_inc     | cla         | width=6        | 0.0 | 13654 |
| DW02       | DW02_mult    | csa         | A_width=3      | 0.0 | 31452 |
|            |              |             | B_width=3      |     |       |

================================================================================
```

Queries can be substantially more complicated, as in the following example:

```
dc_shell-t> report_cache -dir ~ -design_lib DW02 -mod *mult -accessed_since 21
-param [list {n=(3;100), m=(9;100)} {n=(6; 100), m= (6;100)}]
```

4.6.5 Cache Variables

Cache-related variables and their definitions are provided below.

`cache_read : space separated list`

Defines the list of cache directories that are searched for a part. If the variable is set to an empty list ({}), no models are read from the disk, which effectively turns off caching. By default, `cache_read` is set to the system cache (under `$SYNOPSYS/libraries/syn`) and to your home directory.

`cache_write : string`

Defines the cache directory where optimized parts are written (if not already present). By default, `cache_write` is set to your home directory.

`cache_read_info : Boolean`

Prints a message when a cache element is read, if the value is true. The default value is `false`.

`cache_write_info : Boolean`

Prints a message when a cache element is written out, if the value is true. The message notifies you that a new element was created. The default value is `false`.

`synlib_optimize_non_cache_elements : Boolean`

Determines whether or not to optimize new implementation models. When a required implementation model is not found in the cache, the model is created and, possibly, optimized. When

synlib_optimize_non_cache_elements is true, these new models are optimized. When it is false, the new models are not optimized. The default value is true.

Using unoptimized models decreases the quality of results of timing-driven resource sharing and implementation selection.

```
cache_file_chmod_octal : string
cache_dir_chmod_octal : string
```

Sets cache file and directory permission mode bits. When cache files are created, their mode bits are set to the value of cache_file_chmod_octal. When cache directories are created, their mode bits are set to the value of cache_dir_chmod_octal. The value of both variables is a string that can be translated to an octal number.

Because separate variables are used for directories and for files, the sticky bit can be set specifically on a directory. The sticky bit is the permission mode bit that restricts your ability to delete other users' files. If the sticky bit on a directory is set and if you have write permission on the directory, you can write your own files in the directory; however, you cannot delete other users' files in that directory.

Many UNIX systems allow a sticky bit to be set on directories (setting the sticky bit on files has a very different meaning and is not allowed for cache files). The sticky bit is important because caches are often shared among different users. For more information, refer to the UNIX man page on sticky(8).

The default for cache_file_chmod_octal is 666. The default for cache_dir_chmod_octal is 777. If you want to use a sticky bit for cache directories, set cache_dir_chmod_octal = 1777.

4.6.6 Tips for Improving Use of the Cache

To make better use of the synthetic library cache, you can:

- "Tip 1. Share the cache" on page 52
- "Tip 2. Fill the cache with commonly used IP" on page 53
- "Tip 3. Set variables to improve compile speed" on page 53

4.6.6.1 Tip 1. Share the cache

One way to manage the disk effectively is to share a single cache among several users. Sharing a cache economizes space and compile time by eliminating the duplication of both model storage and model creation.

The main concern when sharing caches is that users can delete other users' cache elements. You can avoid this predicament by setting the sticky bit on the cache directory (refer to the cache_dir_chmod_octal variable under "Cache Variables" on page 51).

Design teams using a small number of ASIC libraries, with only one version of each ASIC library and of the Synopsys software, can share a single cache. In such cases, the whole team can have read and write permission for the cache. Each team member should have lines similar to the following in the ~/.synopsys_dc.setup file:

```
cache_file_chmod_octal = 664
cache_dir_chmod_octal = 775
cache_read = cache_read + group_area
```

```
cache_write = group_area
```

`group_area` is the UNIX file directory where all the parts will be stored.

This setup allows the Synopsys tools, when started up by any member of the team, to cache parts in `group_area`. It also allows members of any other group to read the parts that have been cached.

4.6.6.2 Tip 2. Fill the cache with commonly used IP

The design team leader can use `create_cache` to set up a collection of customized synthetic parts required for the current project, and place them in a (common) cache directory.

4.6.6.3 Tip 3. Set variables to improve compile speed

Another aspect of cache management is setting the cache-control variables to achieve fast execution of the `compile` command. If you are exploring many different design alternatives, you can set the `synlib_optimize_non_cache_elements` variable to `false`; new implementation models added to the cache will not be optimized. The advantage is faster compilation; the cost is lower quality of results from timing-driven resource sharing and implementation selection.

Even when `synlib_optimize_non_cache_elements` is `false`, Design Compiler can retrieve optimized cache elements when these exist in a directory listed in your `cache_read` variable.

Later in the design process, when you are repeatedly compiling the same design or variations of the same design, you can improve your timing-driven resource sharing and implementation selection results by setting `synlib_optimize_non_cache_elements` back to `true`. Only the first `compile` requires the overhead of optimizing a given cache model.

4.7 Other minPower Reports

4.7.1 Reporting Delay and Power Contribution

- **Command name:** analyze_dw_power -- lists the slack and power contribution of synthetic cells in the current design.

For more information on this command and its use, see "Reporting Delay and Power Contribution" on page 71, or refer to the analyze_dw_power command man page.

4.8 Summary of Advanced Features

1. You can manually select modules and implementations by including directives in your HDL design description, or by reading your design into `dc_shell-t` and issuing the `set_implementation` command.
2. You can disable high-level optimization of synthetic operators by using the `replace_synthetic` command.
3. You can disable specific synthetic modules and their implementations with the `dont_use` command.
4. You can establish priorities among the implementations of a given module with the `set_implementation_priority` command.

5. Implementations are instantiated by default as a level of hierarchy. You can ungroup these levels with the `set_ungroup` command.
6. You can delete unconnected ports from selected cells in your design by using the `remove_unconnected_ports` command.
7. You can improve your use of the Synthetic Library cache by:
   - sharing the cache
   - filling up the cache with commonly used parts
   - Improving compile speed when you are exploring many different design alternatives by setting the `synlib_optimize_non_cache_elements` variable to `false`

5 Using Licensed Implementations

You must have a license key to use a licensed DesignWare minPower Components. Design Compiler automatically checks out the required licenses when you execute the `compile` command. Thus, most licensing operations are transparent. However, the implementations selected by Design Compiler depend on which licenses are available.

Using designs with licensed implementations involves:

5.1 Basic Licensing Rules and Guidelines

5.1.1 Overview

The DesignWare-LP license feature is required to use DW_lp_* components. Here is a quick overview of the license requirements for DesignWare minPower Components and their license relationship to DesignWare Building Block and Datapath IP.

- The **DesignWare-LP** license does not require the **DesignWare** license as a prerequisite. Building Block sub-components used within DW_lp_* components are licensed with the **DesignWare-LP** feature.
- The DesignWare minPower Components (DWmP) synthetic library is named **dw_minpower.sldb**.
- The traditional "foundation" DesignWare Building Blocks (DWBB) synthetic library is still named **dw_foundation.sldb**.
- To utilize the DesignWare minPower Components, the **DesignWare-LP** license is required.
- To utilize traditional DW foundation, **DesignWare** license is required.

- DC Ultra is required to use the **dw_minpower.sldb** synthetic library and minPower Components. If you attempt to include them in a DC Expert flow, DC issues an error and aborts the compile.

5.1.2 DesignWare-LP License Usage Model

The following describes the usage model for the minPower Components and DesignWare-LP license.

- In the DC Expert flow, minPower Components flow is not supported. If a user adds the dw_minpower.sldb in the synthetic_library/link_library list, DC issues an error message and aborts the compile.
- If dw_minpower.sldb is not added to the synthetic_library/link_library list, the minPower Components flow will not be used.
- Licenses will be checked out at the beginning of the compile. If dw_minpower.sldb is in the synthetic library list, a DesignWare-LP license is checked out regardless of whether or not minPower Components are used.

Note: These flows assume a Power Compiler license (PwrC) has been checked out when needed. If PwrC license is not checked out, DC will abort if the design has power constraint, or Datapath Gating is enabled.

5.1.3 License Management in Other Commands

The following list describes commands behavior that affects license usage:

- **`create_cache`**: This command checks out the required license for the DesignWare component/implementation for which the cache is created. For example, a user creates a cache for module A, implementation B. If a DesignWare-LP license is required by B, then DesignWare-LP is checked out. If a DesignWare license is required by B, then DesignWare is checked out. If B is a free implementation, then neither license is checked out.
- **Read/write ddc design, including DDC cache**: If the design requires both DesignWare and DesignWare-LP license, only check out DesignWare-LP license. If the design requires either DesignWare or DesignWare-LP license, only check out the first license in the string.
- **set_design_license**: The DesignWare-LP license is be treated like other individual licenses. The set_design_license command does not checkout a license. It only set the license string on the design. Currently a design can have “duplicated” license {DesignWare and DesignWare-Foundation-Ultra and DesignWare-Foundation}. The read command will only checkout the super license.
- **set_dont_use_license**: The DesignWare-LP is treated the like other individual licenses.
- **get_license**: DesignWare-LP is treated the same way as other individual licenses. Currently, you can get DesignWare, DesignWare-Foundation, and DesignWare-Foundation-Ultra at the same time using get_license command.
- **remove_license**: DesignWare-LP is treated like other individual licenses.
- **wait_for_license**: DesignWare-LP is treated like other individual licenses.
- **synlib_prefer_ultra_license**: DesignWare-LP will not affect this variable.

5.2 Displaying License Requirements of Implementations

To find out which implementations declared in a synthetic library are licensed, and which keys you need in order to access them, use the `report_synlib` command as shown in the following example.

```
dc_shell-t> report_synlib dw_minpower.sldb
****************************************
Report : library
Library: dw_minpower.sldb
Version: C-2009.06
Date   : Fri Jul 3 17:50:28 2009
****************************************

Library Type            : Synthetic
Tool Created            : C-2009.06
Date Created            : 06.16.09
Library Version         : C-2009.06:C-2009.06-DWmP_0916

Synthetic Modules: ....

Module Implementations:

     Attributes/Parameters:
       v - verify_only
       V - verification implementation
       u - dont_use
       r - regular_licenses
       l - limited_licenses
       d - design_library
       s - priority_set_id
       p - priority
       leg - legal


Module           Implementations      Attributes/Parameters
-------------------------------------------------------------------

DW_lp_piped_div  rtl                  r = DesignWare
                                      d = DW
                                      leg = "(SH_width>=1) && (A_width>=2)"
...
```

According to the above example, the synthetic library dw_minpower.sldb includes the synthetic module DW_lp_piped_div, with `rtl` as one of the implementations.

The `rtl` implementation is the synthesizable implementation that is used to generate hardware. The `rtl` implementation of this component requires a DesignWare-LP license.

If the DesignWare-LP license is available, you can include the `rtl` implementation in your design. You can compile a design that contains the implementation and write the resulting netlist to any Design Compiler output format.

5.3 Displaying the License Status of a Design

You may want to see what licenses are required by your design. You can display this kind of license information in several ways.

5.3.1 Displaying License Information on Designs in the Hierarchy

To find out more information on the current license status of all designs in the hierarchy, use the `report_hierarchy` command. The following example uses the `report_hierarchy` command to display the license status of the current design.

```
dc_shell-t> report_hierarchy

****************************************
Report : hierarchy
Design : top
Version: C-2009.06
Date   : Fri Jun 12 20:07:58 2009
****************************************

Attributes:
    r - licensed design

top
     bottom                  r
     ...
1
```

According to the attribute information, the design `top` requires no licenses but the design bottom is licensed.

Designs that contain unmapped IP are sometimes reported as limited designs, even though you may have a regular license for the IP in question. The purpose of this limitation is to keep the proprietary internal structure of some IP from view. By compiling such a design, however, you turn it into a regular design.

5.3.2 Displaying License Information on a Specific Design

Although the `report_hierarchy` command displays the current status of the designs, the command does not show you which licenses are required for a design and which licenses give you full access to a design. To display this information, use the `report_design` command. In the following example, the current design is changed from `top` to `bottom`, and the license status of bottom is displayed with the `report_design` command.

```
dc_shell-xg-t> current_design bottom
Current design is 'bottom'.
{bottom}

dc_shell-xg-t> report_design

****************************************
Report : design
Design : bottom
Version: C-2009.06
Date   : Fri Jun 12 20:17:42 2009
****************************************

...
Required Licenses:

    DesignWare-LP

...
1
```

The regular DesignWare-LP license gives you full access to the design bottom.

5.4 Excluding Licensed Implementations

During compilation, Design Compiler normally checks out the available licenses needed to build your design. Design Compiler does not consider implementations that require licenses you do not have.

In some instances, you may need to exclude specific licenses from being checked out by Design Compiler. In these cases, you can use the variables `synlib_disable_limited_licenses` and `synlib_dont_get_license` to exclude unwanted licenses.

Attention If you exclude the DesignWare-LP license, but have included dw_minpower.sldb in the Library list, or your design contains DW_lp_* minPower Components, the DC compile will abort.

5.4.1 synlib_disable_limited_licenses Variable

When an implementation with a limited license is used to build a design, the design cannot be written out. Because of this limitation, you may not want to use limited licenses. By default, the `synlib_disable_limited_licenses` variable is set to `TRUE`, which restricts Design Compiler from checking out any limited licenses.

5.4.2 synlib_dont_get_license Variable

If you do not want to use a particular license key (for instance, because others have a more critical need for a key in short supply), you can use the variable `synlib_dont_get_license`. Implementations requiring the keys listed in this variable are not used.

Suppose, for example, that the following implementations of an adder module have been installed at your site:

Implementation	License Required
impl0	None
impl1	VERY FAST ADD
impl2	FAST ADD or VERY FAST ADD

If you do not want to use the `VERY_FAST_ADD` license, you can set the `synlib_dont_get_license` variable to

```
synlib_dont_get_license = { VERY_FAST_ADD }
```

If you then use the adder module in a design, Design Compiler will access all the implementations it can. `impl0` does not require any licenses to be checked out, so Design Compiler can freely access the implementation. `impl1` requires the `VERY_FAST_ADD` license, so Design Compiler—given the current setting of `synlib_dont_get_license`—cannot access it. Design Compiler accesses `impl2` by checking out the `FAST_ADD` license.

5.5 Wait for Design License

During a compile, Design Compiler will, by default, exit from the process if there are no DesignWare licenses available. That is, if your site has a valid DesignWare license key, but the license is not available (the key is checked out by another user), DC will issue an error message and exit from the command.

5.5.1 synlib_wait_for_design_license Variable

Some designers prefer to have DC wait until the required key is available, then resume the process. This is done by setting the `synlib_wait_for_design_license` variable.

The default value of the `synlib_wait_for_design_license` variable is an empty list. When the variable is set to an empty list, DC will behave normally. If it is set to a list of design license names, then the DC's compile, read, elaborate commands will wait for one of the necessary design licenses to become available rather than aborting the process.

To cause Design Compiler to wait for a DesignWare-LP license to become available before proceeding with the compile, set the `synlib_wait_for_design_license` variable to:

```
dc_shell-t> set synlib_wait_for_design_license [list "DesignWare-LP"]
```

5.5.2 Excluding Unavailable Licenses

If implementations are authorized for a site, but licenses are not currently available, the compile command generates an error message and aborts:

```
dc_shell-t> compile
Error: The synthetic library part implementation
       'dw_lp_piped_div' should be available for use during the
       compile command, but the implementation is not
       enabled because all of the following regular
       licenses have been checked out:
       DesignWare-LP
       The design compiler command is being aborted
       because the missing implementation may affect
       compile results. (SYNL-16)
Information: Compile terminated abnormally. (OPT-100)
Current design is 'top'.
0
```

If you want to continue, you can use the `synlib_dont_get_license` variable to exclude unavailable licenses. Note that the quality of your results can suffer because optimal implementations may not be considered.

5.6 Checking Out Licenses Manually

Although Design Compiler automatically checks out the required licenses during compilation, you may want to check out licenses manually before that.

In the report listed in the design `bottom` requires the DesignWare license. You may want to reserve this license (someone else may be using it) before you compile the design. To get a license, use the `get_license` command. For example, to get the DesignWare license, enter the following statement (the `list_licenses` command verifies that the license has been acquired):

```
dc_shell-t> get_license DesignWare-LP
1
dc_shell-t> list_licenses
          DesignWare-LP
1
```

Suppose a design requires either a limited license or a regular license, and you have acquired the limited license. You may want to check out the regular license instead. Use the `get_license` command to check out a regular license; the design is automatically converted from a limited design to a regular design.

5.7 Ungrouping Licensed Implementations

Ungrouping a licensed design causes the parent design to inherit license information. When this situation occurs, a warning message is displayed, notifying you that license information has changed. In the hierarchy report in the following example, the design `bottom` requires a regular license. The parent design, `middle`, requires none.

```
dc_shell-xg-t> report_hierarchy

****************************************
Report : hierarchy
Design : top
Version: C-2009.06
Date   : Fri Jun 12 20:07:58 2009
****************************************

Attributes:
    r - licensed design
top
     bottom                  r
     ...
1

dc_shell-xg-t> ungroup -all
Warning: Design 'top' inherited license information from design 'middle'. (DDB-74)
1
```

After the subdesign bottom is ungrouped into the design `top`, `top` now requires the same licenses that `bottom` required.

Grouping a design similarly causes license information to be passed on from the parent design to the newly grouped design.

5.8 Summary of Use of Licensed Implementations

1. Most licensing operations are transparent.
2. You can display information on licensed implementations by using the `report_synlib` command on a synthetic library.
3. You can display license information on designs in a hierarchy by using the `report_hierarchy` command.
4. You can display license information on specific design by using the `report_design` command.
5. Some designs are read as limited because licensed parts have not yet been mapped. Once you map the parts, the designs are no longer limited.
6. You can prevent Design Compiler from checking out specific licenses by setting the following variables:
   - `synlib_disable_limited_licenses`
   - `synlib_dont_get_license`

   Site authorization also limits the licenses that Design Compiler can check out.
7. You can check out licenses manually by using the `get_license` command.
8. Grouping and ungrouping designs leads to changes in license status. License information is inherited or passed on.

A
minPower Application Notes

This appendix gives you the latest application information, to help you squeeze the most power from your designs using DesignWare minPower Components. Because all designs are different and the benefit of using DesignWare minPower Components can vary, this section is provided to give additional "application" techniques to help you tune your design for low power operation.

The following topics are contained in this Application Notes appendix:

A.1 minPower Setup and Flow

Qualify: use the `analyze_datapath` and `report_resources` commands to determine if your design is a good candidate for minPower optimizations.

Setup: add dw_minpower.sldb to `synthetic_library`, `link_library`:

```
set synthetic_library "dw_foundation.sldb dw_minpower.sldb"
set link_library [concat  $link_library $synthetic_library]
```

Flow:

- Annotate design with switching activity from RTL simulation before synthesis (SAIF file):

  ```
  read_saif -input <rtl>.saif
  ```

- Compile the design:

  ```
  compile_ultra
  ```

A.2 minPower Benefit Guidelines

The following guidelines and optimizations can help you fine-tune your minPower optimization.

A.2.1 Dynamic Power

- Most dynamic power optimizations implemented in the datapath generators are only effective in large blocks, in particular large sum-of-products or product-of-sums.
- If a design contains mainly small blocks and only few multipliers (so mostly adders, comparators, selectors) only small dynamic power gains can be expected.
- Several optimizations are base on transition probabilities (switching activities), therefore it is important that realistic activities are present at datapath inputs before synthesis. The default activities assigned by Power Compiler are not always appropriate. Switching probabilities can be achieved in two ways:
  - Assign application specific switching activities to the primary inputs:

    ```
    set_switching_activity -static_probability 0.5 -toggle_rate 0.2 -period $period in1
    ```
  - Annotate switching activities from RTL simulation using SAIF files. This approach is preferred because detailed activity is obtained and more ports get annotated.

    ```
    read_saif -input rtl.saif
    ```

A.2.2 Leakage Power

- Multi-Vt libraries are required to optimize leakage power.
- Leakage power improvements are possible in any design, but improvements are more likely in designs with loose constraints.

A.2.3 Glitching Power

- Glitching power is the extra power consumed by glitches (incomplete transitions or transitions that are undone later in the same cycle) and is part of the dynamic power. Glitching power can be very significant in datapath circuits (up to 60% of the total power in large multipliers).
- Glitching power is currently not optimized nor reported by Power Compiler.
- Measuring glitching power requires that a timing simulation be performed, and back-annotation file of switching activities (.saif file) for power reporting.
- Glitching power reduction is difficult and can increase dynamic base power (dynamic power without glitching power).
- Glitching optimization techniques are:
  - Delay balancing: prevents the generation of glitches, needs to consider the back-end.
  - Architecture selection: for example, a non-Booth multiplier has less glitching than a Booth multiplier. See “Architecture Selection” on page 66.
  - Special cells: See “Special Cells” on page 66.

A.3 Specific minPower Optimization Techniques

The following techniques can help you optimize your design, relative to specific design characteristics.

A.3.1 Datapath Gating

One of the features of DesignWare minPower is built-in datapath gating. The datapath architectures are designed to be datapath gating friendly enabling users to turn off switching in the entire datapath block when output is not required, with no timing overhead and minimal area overhead. DG optimizes datapath structures to identify optimal operands to gate and merges datapath gating logic with computation logic, eliminating the requirements for external isolation gates. And even if timing is not met, minPower has the capability to perform bit-wise incremental optimization on non-critical fan-outs only and still give power savings.

NOTE: The Operand Isolation feature has been obsoleted and replaced with this new built-in Datapath Gating (DG) feature. To use the DG feature, you need to set the following variable:

```
set power_enable_datapath_gating true
```

The DG feature is available from 2010.03-SP2 onwards and needs a DesignWare-LP license. You can report the status of datapath gating for your design by using report_datapath_gating command. This tells you how many operators were present, how many were gated, and how many were not. For those operators that are not gated, you can use `report_datapath_gating -ungated` option to report the reasons why these operators were not gated.

A.3.2 Transition-Probability-Based Adder Tree

Description: The carry-save adder tree in multipliers or vector summers is optimized based on transition probabilities (switching activity). Activity and dynamic power in the adder tree is reduced.

Switch: `set_dp_smartgen_options -tp_opt_tree auto` (default: auto)

Benefit:

- Multipliers and vector summers (more benefit for larger blocks).
- Loose constraints only, i.e. timing is easily met (reason: optimization increases delay).
- Non-uniform activity profiles at inputs (SAIF annotation prior to synthesis recommended).
- Applications: Signal processing (input values with low magnitude, higher bits switch rarely).

A.3.3 Transition-Probability-Based Operand Selection

Description: Selects the input operand with lower activity to be connected to the Booth encoding in multipliers. Less logic with high activity results in lower dynamic power.

Switch: `set_dp_smartgen_options -tp_oper_sel auto` (default: auto)

Benefit: Large multipliers.

- The two inputs have significantly different switching activities (SAIF annotation prior to synthesis recommended).

Applications: Filters with variable coefficients (coefficient input changes rarely).

A.3.4 NAND-Based Multiplier

Description: Partial-product bits are generated with NAND gates instead of AND gates (or Invert-NOR). NAND gates are often slightly more efficient and the lack of inverters can reduce glitching.

Switch: `set_dp_smartgen_options -mult_nand_based auto` (default: false)

Benefit: Dynamic and glitching power.

Application: Any non-Booth multiplier.

Note: Currently turned off by default because of Formality has potential bad logic issues.

A.3.5 Radix-4 non-Booth Multiplier

Description: Radix-4 partial-product generation for non-Booth multipliers reduces the size of the adder tree to save area and power.

Switch: `set_dp_smartgen_options -mult_radix4 auto` (default: auto)

Benefit: Reduced area and power in small to medium sized unsigned multipliers (up to 24-bit input width).

Application: Non-Booth multipliers with input widths up to 24 bits.

A.3.6 Architecture Selection

Description: During datapath generation, different circuit architectures are evaluated and the best architecture is selected based on constraints (dynamic power is included in the cost).

Switch: `set synlib_dwgen_smart_generation true` (default: true)

Benefit:

- Dynamic and glitching power.
- Biggest benefit in large multipliers (non-Booth vs. radix-4 Booth vs. radix-8 Booth).
- Small benefit possible in small blocks (adders, small multipliers, etc.).

A.3.7 Special Cells

Description: The use of special datapath cells can reduce dynamic power. Supported special cells are 42-compressors, Booth encoder/selector cells (mux-based and xor-based).

Switch: `set_dp_smartgen_options -4to2_compressor_cell|booth_cell|booth_mux_based auto`

Benefit:

- Large multipliers.
- Technology libraries with efficient special cells.
- Special cells can also reduce glitching power (balanced delays produce less glitching, long delays filter out glitches).

A.3.8 Leakage Power

Description: In multi-Vt libraries, leakage power is reduced by mapping to high-Vt (low leakage) cells wherever possible.

Switch: `set_max_leakage_power 0` (automatically turned on through leakage power constraint)

Benefit:

- Technologies with high-Vt cells (multi-Vt library).
- Biggest benefit for loose constraints (reason: high-Vt cells are slower).
- Benefit possible in any kind of datapath (adders, multipliers, small and large blocks).

A.4 Debugging Your minPower Runs

DW minPower benefits most datapath designs. If you don't see power improvement with minPower, check the following.

A.4.1 Setup and Environment

- Check that you are pointing to the correct license. “list_license” should list “DesignWare-LP”.
- Check DC-Ultra version. The first official release version of minPower was 2009.06-SP1.
- The official library name for minPower is: dw_minpower.sldb

A.4.2 Datapath Extraction: Interpreting DW Reports

To benefit from DesignWare minPower Components, there must be adequate datapath content in your design and it must be coded to enable efficient extraction in to datapath block. The `report_resources` command gives a detailed report on the datapath components that were synthesized, including function and operands width/types. Here is a sample report from the `report_resources` command.

```
Resource Report for this hierarchy in file ./test.v
=============================================================================
| Cell            | Module          | Parameters | Contained Operations |
=============================================================================
| mult_x_82_1     | DW_mult_tc      | a_width=8  | mult_82              |
|                 |                 | b_width=8  |                      |
| DP_OP_6_296_894 |                 |            |                      |
|                 | DP_OP_6_296_894 |            |                      |
=============================================================================
```

The report shows the following:

- A Resource Report for arithmetic operators extracted from RTL code that are mapped to individual DesignWare components, for singleton components (that is, individual operations implemented by a discrete DesignWare component) and complex datapath blocks.

- A Datapath Report for arithmetic operators that are merged into a single datapath block by datapath extraction. This includes contained RTL operations, interface (input and output ports) and functionality. Internal ports are in carry-save format whenever possible/beneficial.

```
Datapath Report for DP_OP_6_296_894
=============================================================================
| Cell                  | Contained Operations                              |
=============================================================================
| DP_OP_6_296_894       | mult_80 sub_81 add_81                             |
=============================================================================

=============================================================================
|          |      | Data     |       |                                      |
| Var      | Type | Class    | Width | Expression                           |
=============================================================================
| I1       | PI   | Signed   | 8     |                                      |
| I2       | PI   | Signed   | 8     |                                      |
| I3       | PI   | Unsigned | 1     |                                      |
| I4       | PI   | Unsigned | 1     |                                      |
| I5       | PI   | Unsigned | 16    |                                      |
| T0       | IFO  | Signed   | 16    | I1 * I2                              |
| T1       | IFO  | Signed   | 16    | $signed(1'b0) - T0                   |
| T3       | IFO  | Unsigned | 16    | { I3, I4 } ? T1 : T0                 |
| O1       | PO   | Unsigned | 16    | T3 + I5                              |
=============================================================================
```

- An Implementation Report for each arithmetic cell. 'str' is the generic name for the flexible SOP/POS implementation that is used for all complex datapath blocks.
- For singletons, the corresponding implementation name is reported (see datasheets).
- Only implementations of hierarchical modules that are still present in the design are reported.

```
Implementation Report
============================================================================
|                  |                  | Current        | Set            |
| Cell             | Module           | Implementation | Implementation |
============================================================================
| DP_OP_6_296_894  | DP_OP_6_296_894  | str            |                |
| mult_x_82_1      | DW_mult_tc       | pparch         |                |
============================================================================
```

When analyzing your reports, look for the following characteristics to determine if the design is a good candidate to benefit from minPower.

- Large datapath content in the design with large singleton components and complex datapath blocks. Large width singleton components are more likely to benefit from minPower.
- Large multipliers (for example, 16x16) will see big benefits with DW minPower
- Complex datapath blocks with many operators extracted into the block will see a difference in power with minPower
- Datapath content that is not separated by hierarchical boundaries will be better extracted in to large datapath blocks. The larger the extracted block, more optimizations that are possible and hence the better the QoR (timing, area and power).
- Datapath content that is not manually pipelined by insertion of registers between datapath content. Use the retiming feature in DC-Ultra instead. Refer to the DC-Ultra User Manual for more details.

A.4.3 Dynamic Power

In most cases, designs with the previously outlined characteristics will benefit from minPower. If the comparison of your baseline and minPower runs does not show an improvement to dynamic power, the following pointers will help you analyze why.

A.4.3.1 General Dynamic Power Flow

1. Focus on blocks with the biggest datapath content in the design (`report_area -designware`).
2. Use `report_power -hier` to determine if these blocks with large datapath content are contributing to the largest power in the design.
3. Are the DW components in the timing critical portions of your design? (`report_timing`)? If the timing is critical on the datapath blocks, there is little flexibility for minPower to generate architectures suitable for lower dynamic power.

A.4.3.2 Switching Activity

DW minPower uses switching activity to generate power aware architectures. Use RTL SAIF if available or the command `set_switching_annotation` to annotate switching activity in your design.

1. Examine the switching activity on input ports. For example, a report may look like this:

```
DW port: I1
pin: I1_5, net: a[5], sp: 0.5000, tp: 0.0667
pin: I1_4, net: a[4], sp: 0.5000, tp: 0.0667
```

2. If the switching activity of a DW block has sp=0 or tp=0, minPower cannot generate switching or glitch aware architectures, as there is no switching. In this case, trace back to see why the switching on the input is a zero.
3. Look for any warnings and/or error messages from PwrC (`PWR-*`).
4. Use `report_power -analysis_effort high` to increase the simulation vectors for a better sampling rate when there are issues related to low switching activity.
5. Use `report_resources -context` to show switching. For details of this command, see "report_resources" on page 95.

A.4.3.3 Datapath Gating

If the switching activity at the input of the DW block is not zero, and you still do not see the DW blocks datapath gated, the following will help you debug the cause.

1. Use `report_datapath_gating` to determine percentage of the DW blocks datapath gated.
2. Look for any (PWR-59) messages related to the datapath gating flow.
3. To debug datapath gating related issues, use `report_datapath_gating -ungated`

A.4.4 Leakage Power

For DW minPower to generate leakage power optimized structures, the flow should involve a good mix of multi Vt cells in different drive strengths.

1. Set group attributes on each set of Vt cells. This will help when reporting Mult Vt cell percentages used in the design.

```
set_attribute [get_lib_cells tcbn65lphvtwc/*]
threshold_voltage_group HVT -type string
set_attribute [get_lib_cells tcbn65lpwc/*]
threshold_voltage_group NVT -type string
set_attribute [get_lib_cells tcbn65lplvtwc/*]
threshold_voltage_group LVT -type string
```

2. Report the percentages of VT cells using `report_threshold_voltage_group`
3. If the DW blocks are in the critical timing path, minPower is less effective in generating leakage optimized datapath structures.

A.4.5 Library

Just like in synthesis flows, the availability of a good library is important for DW minPower to generate optimized structures. Some problems observed when working with customer libraries are listed below.

1. If you see power degradation or no improvement with minPower, despite the design meeting the qualifications listed above, examine the internal power of the cells in the library. Erroneous and negative values have sometimes been observed for the internal power of cells in certain customer libraries. Since a different mix of cells may be selected in a minPower run than the baseline run, if the cells have negative or zero internal power in the baseline run but not in the cells used in the minPower run, this will manifest as a degradation or no improvement to power.
2. A good mix of multi Vt cells in different drive strengths is important for good synthesis results. Use the `analyze_minpwr_library` command (see "analyze_minpwr_library" on page 91).
3. Libraries with a good selection of datapath cells will benefit even more from minPower. See:

   https://solvnet.synopsys.com/retrieve/004286.html for more information.

A.4.6 Reporting Delay and Power Contribution

The **`analyze_dw_power`** command lists the slack and power contribution of synthetic cells in the current design.

Syntax: **integer analyze_dw_power [-nosplit] [-hierarchy]**

For the generated DesignWare netlist, the slack reported is the best slack that the generator could achieve. For static designs, the command only reports paths with negative slack.

This report also reports the power contribution of all synthesis cells in the current design.

For a full description of this command, see the command man page.

For hierarchical DesignWare components, only the slack and power contribution of the top level design are reported. The imbedded hierarchy is not reported.

Note: To use this command, you must first set synlib_enable_analyze_dw_power to 1.

Example:

The following example generates the analyze_dw_power report for hierarchical DesignWare components in the "test" design:

```
prompt> set synlib_enable_analyze_dw_power 1
prompt> analyze_dw_power -hier
```

```
****************************************
Report : dw_slack_power
Design : test
Version: E-2010.12-SP1
Date   : Wed Jan 19 17:19:40 2010
****************************************

DesignWare Power Contribution report
================================================================================
|                          |         |         | dyn     | lkg     |            |        |
|                          | Path    | DW      | power   | power   |            |        |
| Cell                     | group   | slack   | %       | %       | opt mode   | Attr   |
================================================================================
| add_x_29_8 (DW01_add_DG)| default | -19.87 | 72.32   | 61.34   |            | u      |
| DP_OP_257J1_124_2676     | default | -33.95 | 25.73   | 35.80   |            | u      |
================================================================================

Attribute:
            u - ungrouped

Power of detected synthetic parts
---------------------------------
No DW parts to report!

Estimated power of ungrouped synthetic parts
--------------------------------------------
  DW cell count: 2
  Dynamic Power: 5.5592e+00  98.05%
  Leakage Power: 1.0676e-03  97.14%
--------------------------------------------
Total synthetic cell dynamic power:  5.5592e+00   98.05%  (estimated)
Total synthetic cell leakage power:  1.0676e-03   97.14%  (estimated)
```

A.5 Usage Guidelines for Datapath Gating (DG)

This section summarizes usage guidelines addressing Datapath Gating (DG) power optimization. The goal of this section is to achieve the following.

- Utilize automatic DG to get optimal power QoR (Quality of Results) without degrading timing.
- Address potential issues when DG is used in conjunction with other optimization techniques.

A.5.1 Datapath Gating Overview

In a datapath intensive design, the complex arithmetic circuits can contribute to a high percentage of the power consumption of the design. If the outputs of a datapath block are not observable under certain conditions, the circuit's dynamic power can be reduced by adding isolation logic with control signals to hold all or part of the output signals at a constant state when they are not observable. To fully exploit the potential of datapath gating in reducing power consumption, it is important to understand how the RTL coding can influence the results you can achieve. Following these guidelines you can ensure that you get the best quality of results for timing and power in your design using datapath gating.

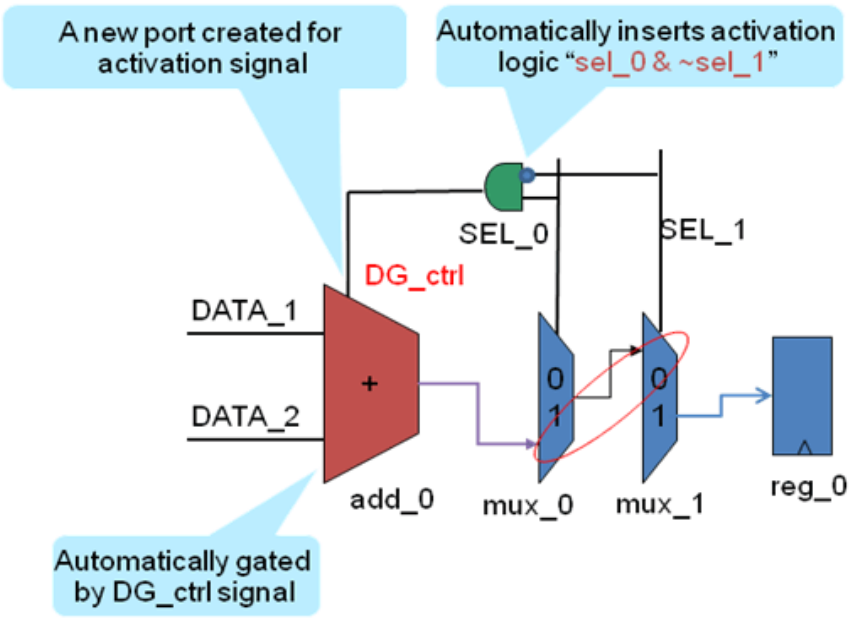
One of the features of DesignWare minPower is built-in Datapath Gating (DG). The datapath architectures are designed to be datapath gating friendly, enabling users to turn off switching in the entire datapath block when the output is not required, with no timing overhead and minimal area overhead. DG optimizes datapath structures to identify optimal operands to gate, and merges datapath gating logic with computation logic, eliminating the requirements for external isolation gates. And even if timing is not met, DG has the capability to perform bit-wise incremental optimization on non-critical fan-outs only and still give power savings.

The goal of this document is to provide designers with a clear understanding of how to achieve the lowest power design using proven structures and techniques. These include how to utilize the automatic features of datapath gating to achieve an optimal power implementation without degrading the timing of the design and how to address potential issues when datapath gating is being used in conjunction with other critical optimization techniques. In other words, this application note explains how to achieve maximum power optimization in your designs using the datapath gating technology for power optimization with minPower technology. A review of the type of circuits that benefit from datapath gating as well as coding guidelines and trade-offs between automatic and manual techniques is discussed that can help you in achieving best QoR for a low power implementation of your design.

A.5.2 Circuits that Benefit from DG

It is important to understand the types of circuits that can benefit from datapath gating. Consider the basic example circuit shown in Figure A-5.

Figure A-5 Design with Datapath Gating



In this example, you can see how datapath gating is used to control when a circuit needs to be active. The operator `add_0` is gated by an activation signal through a newly created port `DG_ctrl` on it. The activation signal is "`SEL_0` & ~`SEL_1`", which is the condition for observing the adder's output on the inputs of register bank reg_0.

A.6 General Guidelines for DG

The ideal candidates for DG are DesignWare datapath blocks that have medium to large complexities. These include both extracted complex datapath blocks and inferred singleton arithmetic operators such as adders, multipliers, and shifters.

To be able to perform datapath gating, the fanout of the combinational circuit needs to have an observability don't care (ODC) condition. If the output of the circuit is always observed, there's no datapath gating opportunity. The following summarizes the conditions for gating a datapath block.

- Its associated arithmetic operation is supported for datapath gating. Currently, all extracted complex datapath blocks and the vast majority of arithmetic singleton operators are supported. For singleton operators, the DW module must be inferred from RTL code, not instantiated. Refer to the DW synthetic library documents for more details.
- The output of the datapath block is not required under certain conditions.
- After applying datapath gating, the total dynamic power consumed by the block and its fanout region is reduced.
- Applying datapath gating will not degrade the timing.
- After applying Datapath Gating, the total dynamic power consumed by the block and its fanout region is reduced.

Important factors that can influence Datapath Gating are:

- The switching activities (static probability and toggle rate) of the activation signal on the `DG_ctrl`. The lower static probability and toggle rate, the more power reduction datapath gating will likely achieve.
- The switching activities on the data inputs.
- The power consumption of the block and its fanout circuit region.
- Critical timing of the `DG_ctrl` signal and the other inputs.

Next are several common RTL coding styles that lead to DG opportunities.

A.6.1 Coding Styles for Successful Datapath Gating

In this section, some of the most common RTL coding styles are discussed that can lead to DG opportunities enabling lower power designs.

A.6.1.1 Conditional assignments driven by arithmetic operators.

In this example, the adder (b+c) can be gated by an activation signal (~EN & ~sel).

Example A-2 Conditional assignments driven by arithmetic operators

```
module test(a,b,c,clk,EN,sel,out);
  input [7:0] a,b,c;
  input clk;
  input EN,sel;
  output [7:0] out;
  wire [7:0] comb_wire;
assign comb_wire = sel ? a : (b+c);
assign out = EN ? 8'b0 : comb_wire;
```

```
endmodule
```

A.6.1.2 Conditional Capture at Registers Inside Processes.

There are several coding styles that allow conditional capture conditions at registered boundaries inside of a process. The most common include an if-else structure, case statements and bit-wise logic. Each condition is show in the following examples.

Example A-3 if-else statement

```
module test(a, b, SEL_0, SEL_1, reg_o, clk);
input [7:0] a, b;
input SEL_0, SEL_1, clk;

output [7:0] reg_o;
reg [7:0] reg_o, tmp, add_0;

always @(posedge clk)
begin
  add_0 = a + b;
  if (SEL_0) begin
    tmp = add_0;
  end else
    tmp = 8'b0;

  //conditional capture at the register input
  if(!SEL_1) begin
    reg_o = tmp;
  end
end

endmodule
```

Example A-4 Case statement

```
module test(a, b, SEL_0, SEL_1, reg_o, clk);
input [7:0] a, b;
input SEL_0, SEL_1, clk;
output [7:0] reg_o;
reg [7:0] reg_o, tmp, add_0;

always @(posedge clk)
begin
  add_0 = a + b;
  case({SEL_0, SEL_1})
    2'b10:   reg_o <= add_0;
    2'b00:   reg_o <= 8'b0;
    default: reg_o <= reg_o;
  endcase
end

endmodule
```

Example A-5 Bit-wise logic operation

```
module test(a, b, SEL_0, SEL_1, reg_o, clk);
input [7:0] a, b;
```

```
input SEL_0, SEL_1, clk;

output [7:0] reg_o;
reg [7:0] reg_o, tmp, add_0;

always @(posedge clk)
begin
  add_0 = a + b;
  tmp = add_0 & {8{SEL_0}};
  reg_o = tmp | {8{SEL_1}};
end

endmodule
```

A.6.1.3 Cascaded Selection verse Parallel Selection

When implementing selection logic, the designer can choose either cascaded structure or parallel structure. The following examples implement the same functionality with a cascaded and a parallel selection structure, respectively. As for DG, these two implementations normally achieve the same result because the activation signal is computed globally with respect to the loading registers and the primary outputs. For example, although the local activation signal of `mult_0` is different in the cascaded case (`en_0`) and in the parallel case (`en_0` & `en_1` & `en_2`) its global activation is "`en_0` & `en_1` & `en_2`" for both cases. Nevertheless, the structure of selection logic will affect the ordering of computing the global activation signal. This can have an impact when the global activation signal needs to be truncated to improve the timing. For instance, when `en_2` is late, it may cause timing degradation if it is used to gate `mult_0`. To avoid this issue, minPower can truncate the global activation signal and use "`en_0` & `en_1`" to gate `mult_0`. It is easier to do so with the cascaded structure. Activation signal truncation is not currently supported, but it is planned in the near future. Hence, you should use the cascaded structure when the selection signals can be critical and place the slower selection signals closer to the outputs, if possible.

Example A-6 Cascaded Selection Structure

```
module test(a, b, en_0, en_1, en_2, reg_o, clk);
input [7:0] a, b;
input clk, en_0, en_1, en_2;

output [7:0] reg_o;
reg [7:0] reg_o, mux_0, mux_1;

wire [7:0] add_0, sub_0, sub_1, mult_0;

assign mult_0 = a * b;
assign add_0 = a + b;
assign sub_0 = a - b;
assign sub_1 = a - 2*b;

always @(posedge clk)
begin
  if (en_0) begin
    mux_0 = mult_0;
  end else
    mux_0 = add_0;

  if (en_1) begin
    mux_1 = mux_0;
  end else
```

```
    mux_1 = sub_0;

  if (en_2) begin
    reg_o = mux_1;
  end else
    reg_o = sub_1;
end

endmodule
```

Example A-7 Parallel Selection Structure

```
module test(a, b, en_0, en_1, en_2, reg_o, clk);
input [7:0] a, b;
input clk, en_0, en_1, en_2;

output [7:0] reg_o;
reg [7:0] reg_o, mux_0, mux_1;

wire [7:0] add_0, sub_0, sub_1, mult_0;

assign mult_0 = a * b;
assign add_0 = a + b;
assign sub_0 = a - b;
assign sub_1 = a - 2*b;

always @(posedge clk)
begin
  casex ({en_0, en_1, en_2})
    3'b111: reg_o = mult_0;
    3'b011: reg_o = add_0;
    3'bx01: reg_o = sub_0;
    3'bxx0: reg_o = sub_1;
  endcase
end

endmodule
```

A.6.2 Automatic Gating vs. Manual Gating

The datapath gating performed by minPower within Design Compiler happens automatically without your input on where and how the gating shall be done. On the other hand it is quite common that the designer has added isolation logics in the RTL code to perform manual gating. This manual approach can complement automatic gating by having more control on the locations and methods of gating.

Table A-4 compares two implementations of a bi-directional shifter; one uses manually gating and the other uses automatic gating. In this example, the automatic gating uses a more efficient method to gate the shifters. As a result, both area and power are better with automatic gating.

The following are general guidelines when considering manual gating in the DG flow.

- Whenever possible, run automatic DG first on a design before adding manually gating. For DesignWare blocks that can be gated by both manual and automatic methods, normally automatic gating gives better QoR and more flexibility.

- Use manual gating on a block if automatic gating cannot handle it or find the optimal activation signal. Automatic gating should still be used on the whole design, although it is unlikely to improve the power of manually gated blocks.
- When the design has tight timing constraints, automatic DG may rollback the gating logic in order to improve delay during compile, and thus give up power improvement opportunities. If it is known that the design is able to eventually meet timing, manually gating can be used to assure that the expected power saving is retained.

Table A-4 Manual and Automatic Gating Comparison

Manual	Auto

`input [31:0] a, b;`
`input [4:0] sh;`
`input sel;`
`output [31:0] o;`
 	
`// a, sh manually gated`
`assign rs_gate_a = {32{sel}} & a;`
`assign ls_gate_a = {32{~sel}} & a;`
`assign rs_gate_sh = {5{sel} & sh;`
`assign ls_gate_sh = {5{~sel}} & sh;`
 	
`always @(*) begin`
`if(sel == 1'b1) begin`
`o = rs_gate_a >> rs_gate_sh;`
`end`
`else begin`
`o = ls_gate_a << ls_gate_sh;`
`end`
`end`
	`input [31:0] a, b;`
	`input [4:0] sh;`
	`input sel;`
	`output [31:0] o;`
	
	`// <<, >> gated automatically by DC`
	`if(sel == 1'b1) begin`
	`o = a << sh`
	`end else begin`
	`o = a >> sh`
	`end`

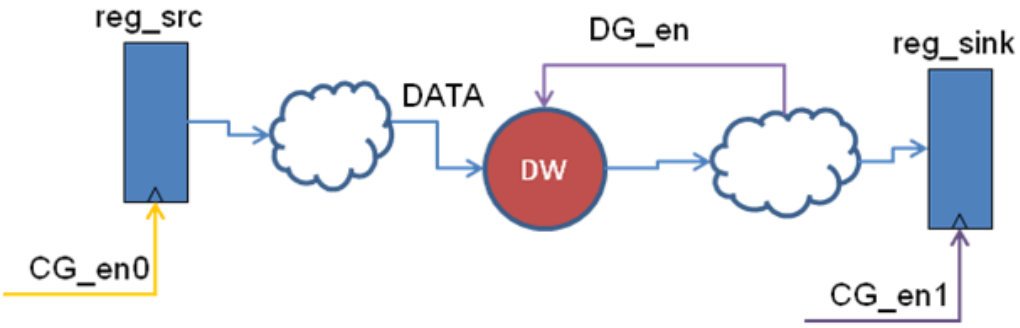
`Dynamic power: 31.81` `Area: 1006`	`Dynamic power: 26.27` `Area: 850`

A.6.3 Datapath Gating with Clock Gating

Clock Gating (CG) is one of the most widely used technique in a power optimization flow. CG shuts down a clock signal when the loading registers are not enabled. In the following discussion we assume that "`compile_ultra -gate_clock`" is used for CG.

In the compile flow, CG insertion is done before DG, and these two optimizations are relatively independent of each other. However, the clock activation signals used by CG and the datapath activation signals used by DG can have a common subset of conditions. In those cases, CG and DG will influence each other. Refer to Figure A-6 where the DG activation signal for the DW block is `DG_en` and the CG activation signal for the source (sink) register bank is `CG_en0` (`CG_en1`).

Figure A-6 Datapath Gating with Clock Gating



To illustrate this, consider the four different scenarios that follow:

Scenario 1: `DG_en == '0''CG_en0 == '0'`.

A special case that meets this condition is `DG_en == CG_en0`. In this case, if the data inputs of the DW block are driven solely by `reg_src` directly or indirectly, then they will remain unchanged when `DG_en == '0'` because `CG_en0 == '0'`. There is no additional reduction in switching activities by DG. Since minPower automatically evaluates if DG can improve the power of a candidate block for gating, it will likely reject DG under such scenarios.

On the other hand, if the DesignWare component is also driven by other registers or primary inputs that still generate switching activities when `DG_en == 0`. Applying CG on `reg_src` will reduce the power saving achievable when applying DG because CG already partially suppresses the activities on the inputs of the block. As a result the DesignWare block is less likely to be gated and, if it is gated, the power saving from DG will be smaller.

A special case is that `DG_en==CG_en0`.

Scenario 2: `CG_en1 == '0' 'DG_en == '0'`.

A special case that meets this condition is `DG_en == CG_en1`. In this case, the DesignWare component output is held constant whenever reg_sink is shut down. This reduces or even stops the net switching activities on the inputs of reg_sink. Therefore, DG provides extra power savings on top of CG.

Scenario 3: `CG_en0` and `DG_en` have a common subset.

When both `CG_en0` and `DG_en are '0'`, the source register bank and the DesignWare component are gated at the same time. Therefore, the total power saving is less than the sum of the savings achievable by CG and DG alone.

Scenario 4: `CG_en1` and `DG_en` have a common subset.

This is similar to **Case 2** except that the sink registers have more input switching activities consuming more power when `CG_en1 == '0'`.

A.6.4 Datapath Gating with Pipelining

A DesignWare component can be pipelined to improve timing. Depending on the target latency, a number of register banks are initially inserted at the output of the block. Then minPower moves the pipeline registers backward into the DesignWare block to achieve the desired performance.

Example A-8 has 3 pipeline stages inserted in the RTL. The multiplier can be gated by signal "`en`".

Example A-8 Multiplier with 3 pipeline stages

```
module test(a, b, en, reg_o, clk);
input [15:0] a, b;
input en, clk;

output [31:0] reg_o;
reg [31:0] reg_0, reg_1, reg_o, mult_o;

always @(posedge clk)
begin
  mult_o = a * b;
  if(en) begin
    reg_0 <= mult_o;
    reg_1 <= reg_0;
    reg_o <= reg_1;
  end
end
endmodule
```

The recommended flow for pipelining consists of the following steps.

- Compile initially with multi-cycle constraints on the blocks to be pipelined.
- Remove the multi-cycle constraints and move the pipeline registers using *optimize_registers*.
- Perform incremental compile(s) to further improve the QoR.

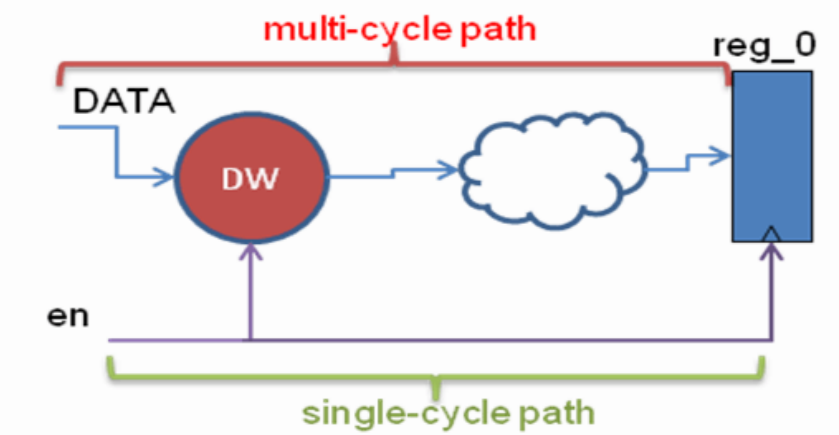
A typical script for Example A-8 is presented in Example A-9. The multi-cycle paths start from the inputs of the multiplier and end at the inputs of the first register bank reg_0. This script works well without DG, but will very likely lead to the rejection of gating on the multiplier in the DG flow. The reason is that connecting signal "en" to the multiplier introduces a single-cycle path group on it and makes the multiplier unable to meet timing constraints. Figure A-7 illustrates the case when "en" is connected by DG.

To avoid this issue, the multi-cycle path constraints must be set to the "en" port for the initial compile.

Example A-9 Pipelining script: multi-cycle paths starting from port a and b, respectively

```
create_clock -name sys_clk -p $period [get_ports clk]
set_input_delay -max 0.0 -clock sys_clk [all_inputs]
set_output_delay -max 0.0 -clock sys_clk [all_outputs]

#set multi-cycle path constraints initially
set_multicycle_path 3 -setup -from [get_ports {a b}]
set_multicycle_path 2 -hold -from [get_ports {a b}]

compile_ultra

remove_constranint -all

#set single-cycle path constraints
create_clock -name sys_clk -p $period [get_ports clk]
set_input_delay -max 0.0 -clock sys_clk [all_inputs]
set_output_delay -max 0.0 -clock sys_clk [all_outputs]

#perform pipelining for the multiplier
optimize_register -no_compile

compile_ultra -incr
```

Figure A-7 Path Groups Resulting from Script 1



A.6.5 Tips for Datapath Gating

With an understanding of the types of structures that are best suited for datapath gating, it is critical to be able to guide the tools used in the designs optimization. The following sections present some of the techniques you should use in your optimization scripts to implement a low power design.

A.6.5.1 Switching activity annotation

To maximize the benefits of DG, it is important to properly annotate switching activities on the design. Here are some recommended practices when performing switching annotation.

- Run RTL simulation to obtain realistic switching activities and annotate them on the primary inputs and registers. Make sure that the special nets are annotated properly, such as clock, set/reset, scan enable, clock enable, etc. For nets that hold constant values during normal function mode, use "`set_case_analysis`" to guide power analysis.
- After annotation, use "`report_saif`" to examine the annotated nets. In general, a higher percentage of annotation leads to more accurate power analysis and better DG results.
- After compile, use "`report_resources -context`" to exam the switching activities on the DesignWare blocks of interest. Example A-10 shows the report output. It is important to check that the data inputs and the enable port (`DG_ctrl`) of the block have reasonable switching activities. A common problem is when many inputs have zero switching activity. This often occurs when the registers driving these inputs are not properly enabled.

Example A-10 Generator Context Report for add_x_8_1

```
============================================================
|    Input |          |                    |              With |
|    Ports |    Width |    Variable Inputs | Switching Activity |
============================================================
|        A |        8 |            8  100% |            8  100% |
|        B |        8 |            8  100% |            8  100% |
|       CI |        1 |            0    0% |            0    0% |
|  DG_ctrl |        1 |            1  100% |            1  100% |
============================================================
```

A.6.5.2 Breaking combinational loops

One of the requirements for DG is that after connecting the activation signal there is no combinational loop introduced. In Example A-11, activation signal `en_0` can be connected to `add_0` for DG. But `en_1` cannot be used for gating because there would be a combinational loop "`en_1(mult_0[0])` --> `DG_ctrl` --> `en_1(mult_0[0])`". Currently minPower always tries to use the global activation signal "`en_0` & `en_1`", which will be rejected due to the combination loop it introduces. Thus, mult_0 will not be gated by any signals.

To avoid this problem, the logic driving `en_1` is duplicated in Example A-12 so that `en_1` can be used to generate gating signal "`en_0` & `en_1`" for mult_0.

Example A-11 `en_1` introduces combinational loops if used for DG.

```
module test(a, b, en_0, reg_o, clk);
input [7:0] a, b;
input clk, en_0;

output [7:0] reg_o;
reg [7:0] reg_o, mux_0;

wire en_1;
wire [7:0] mult_0;

assign mult_0 = a * b;
assign en_1 = mult_0[0];

always @(posedge clk)
begin
  if (en_0) begin
    mux_0 = mult_0;
  end else
    mux_0 = 1'b1;

  if (en_1) begin
    reg_o = mux_0;
  end
end

endmodule
```

Example A-12 Duplicate the logic of "`en_1`" to removed the combinational loops

```
module test(a, b, en_0, reg_o, clk);
input [7:0] a, b;
input clk, en_0;

output [7:0] reg_o;
reg [7:0] reg_o, mux_0;

wire en_1;
wire [7:0] mult_0;

assign mult_0 = a * b;
assign en_1 = a[0] & b[0];

always @(posedge clk)
begin
```

```
  if (en_0) begin
    mux_0 = mult_0;
  end else
    mux_0 = 1'b1;

  if (en_1) begin
    reg_o = mux_0;
  end
end

endmodule
```

A.6.5.3 Discovering Activation Signals

For a given DesignWare block, minPower searches for its activation signals starting from the output ports of its parent module or the data inputs of its loading registers. Therefore, it cannot discover activation signals that are outside its parent module or connected to the MUX operators driven by its loading registers.

In Example A-13, activation signal `SEL` is outside of hierarchy 'sub_mod' which contains operator 'add_0.' So minPower cannot discover this signal and use it to gate `add_0`. To allow DG in this case, the user needs to ungroup the sub-module or merge the external activation signal into it.

Example A-13 Activation signal outside the parent module

```
module sub_mod(a, b, add_o);
input [7:0] a, b;
output [7:0] mult_o;

  assign add_0 = a + b;
endmodule

module test(a, b, SEL, reg_o, clk);
input [7:0] a, b;
input SEL, clk;

output [7:0] reg_o;
reg [7:0] reg_o, add_o;

sub_mod U1(a, b, add_o);

always @(posedge clk)
begin
  if(SEL) reg_o <= add_o;
end

endmodule
```

Example A-14 has two activation signals `en_0` and `EN`. For operator '`add_0`' signal `en_0` can be used for DG while signal `EN` cannot. To explore more DG opportunities, the designer should try to avoid adding selection logic after the loading registers for DesignWare operators.

Example A-14 Activation signal after registers

```
module test(a, b, en_0, EN, out, clk);
input [7:0] a, b;
input clk, en_0, EN;

output [7:0] out;
```

```
reg [7:0] reg_o;

wire en_1;
wire [7:0] add_0;

assign add_0 = a + b;

always @(posedge clk)
begin
  if (en_0) begin
    reg_o = add_0;
  end
end

assign out = EN ? 8'b0 : reg_o;

endmodule
```

B
Qualifying Designs for minPower

This chapter describes the following qualification guidelines and commands:

B.1 Design Qualification Guidelines

The following is a list of guidelines to determine how well your design will benefit from DesignWare minPower Component technology.

B.1.1 Qualification Guideline Checklist

- Check for total Datapath content, should be >= 20% [adders, multipliers, shifters, subtractors]
  - Add `analyze_datapath` DC tcl command after the first compile
  - Large bit-width operators (> 8 bits) will benefit more from minPower
- Check the design timing [`report_timing`] and whether datapath is on critical path
  - Timing not met and datapath blocks are on critical path--design is not qualified.
  - Timing met--design is qualified. Benefit depends on positive slack available for trade-off
  - Timing not met, but datapath blocks are not on critical path (with non-negative slack)--design is qualified. Power benefit depends on positive slack available for trade-off
- Check the power consumption profile of datapath blocks with respect to total power [`report_power -hier -verbose; set compile_ultra_ungroup_dw false`]
  - If contribution is small, expect marginal improvement from minPower
- Check power constraint settings before first compile

- Dynamic or no power constraint [`set_max_dynamic_power 0` or no power const set]
  - Try minPower; improvement for both leakage and dynamic power can be observed
- Leakage constraint [`set_max_leakage_power 0`] -or- both dynamic and leakage constraints [`set_max_dynamic_power 0, set_max_leakage_power 0`]
  - Try minPower only when leakage power is dominant.

- Check if RTL SAIF file is available. If not-
  - use default switching with appropriate activity set for reset and enable signals, or set non-default switching based on application [set_switching_activity]
  - With realistic switching (SAIF), the savings from mP can be maximized [read_saif]
- Check combinational and sequential power for design [`report_power -verbose`]
  - If sequential power is dominant, expect marginal benefit as minPower mainly works on combinational logic

B.2 Best Practices

The following list gives best

- For best QoR (timing and power), avoid hierarchical boundaries and registers in your datapath. Use the retiming feature instead
- Check for settings like smartgen options (opt for speed), don't use on special cells (for example, 4-2 cells) that can restrict QoR benefit.
- Good coding style enables efficient extraction in to large datapath blocks

  https://www.synopsys.com/dw/doc.php/wp/coding_guidelines_aug09.pdf
- Check if any singletons are instantiated, replace with inferred operators for better extraction (if doable)
- minPower performs small trade-offs between area and power; +ve slack and power
  - Trade-offs in timing critical designs are difficult
  - Overconstraining lowers the potential for minPower to optimize for power
- Similar to synthesis, minPower optimizations vary depending on libs/technology nodes
  - A library with more choices in drive strength will likely show more benefit
  - Availability of multi-Vt cells will benefit minPower leakage optimizations
  - Library Guidelines for DesignWare selection reference:

    https://solvnet.synopsys.com/retrieve/004286.html

B.3 Design Analysis Commands

Designware minPower Components provides you several commands and tools to help you determine if your design is a good candidate for power reduction using minPower.

B.3.1 analyze_datapath

Lists the resources and datapath blocks used in the current design.

SYNTAX: analyze_datapath

ARGUMENTS:

`-file`
Provide existing report log file to get the datapath analysis report. The report log file should contain log from `report_resources -hierarchy` and `report_area -designware` commands.

DESCRIPTION:

This command lists the summary of resources and datapath blocks used in the current design. The command displays the area of singleton DesignWare designs and extracted datapath designs. It also summarize the numbers of singleton DesignWare components and datapath designs; the ranges of the designs output width.

EXAMPLES

The following example displays resource information for the current design:

```
prompt> analyze_datapath

************************************************
Estimate of Datapath Content

Extracted Datapath:  area =  45.9%,  parts = 125
Singleton Datapath:  area =  22.4%,  parts = 168

Total Datapath:      area =  68.3%,  parts = 293
************************************************

************************************************
Estimate of Datapath Types

  Note: During compile there may be new operators
  inferred that are not included in the resource report.
  Also datapath components can be added to help meet
  timing constraints. These numbers are estimates.
  Some singletons do not contribute to the potential
  of minPower and therefore are left out in the report below.

Singletons:
                          out width
  component     count   min   max     ave
  ----------------------------------------
  DW01_add         69     7    32    23.5
  DW01_sub         38     9    16    11.5
  DW01_inc         19     9    20    16.4
  DW_cmp            6    12    16    14.8
  DW_mult_uns      35     8    32    19.1
  DW_mult_tc        1    21    21    21.0
  ----------------------------------------


Datapaths:
```

```
                          out width
  operation     count   min   max     ave
  ---------------------------------------
  + -             340     6    32    16.8
  *                50    15    31    22.6
  ?                69     9    32    15.4
  >> <<             0     0     0     0.0
  == !=             5    12    12    12.0
  < <= > >=         0     0     0     0.0

*************************************************
```

The following example shows that the same report is printed out when you use the report log.

```
prompt> report_resources -f resource_area_rpt.txt

*************************************************
Estimate of Datapath Content

Extracted Datapath:  area =  45.9%,  parts = 125
Singleton Datapath:  area =  22.4%,  parts = 168

Total Datapath:      area =  68.3%,  parts = 293
*************************************************


*************************************************
Estimate of Datapath Types

  Note: During compile there may be new operators
  inferred that are not included in the resource report.
  Also datapath components can be added to help meet
  timing constraints. These numbers are estimates.
  Some singletons do not contribute to the potential
  of minPower and therefore are left out in the report below.

Singletons:
                          out width
  component     count   min   max     ave
  ---------------------------------------
  DW01_add         69     7    32    23.5
  DW01_sub         38     9    16    11.5
  DW01_inc         19     9    20    16.4
  DW_cmp            6    12    16    14.8
  DW_mult_uns      35     8    32    19.1
  DW_mult_tc        1    21    21    21.0
  ---------------------------------------

Datapaths:
                          out width
  operation     count   min   max     ave
  ---------------------------------------
  + -             340     6    32    16.8
  *                50    15    31    22.6
  ?                69     9    32    15.4
  >> <<             0     0     0     0.0
```

```
  == !=              5     12     12     12.0
  < <= > >=          0      0      0      0.0

************************************************
```

B.3.2 analyze_minpwr_library

Describes target library information used by power optimization in datapath generation.

SYNTAX: `analyze_minpwr_library`

ARGUMENTS:

`"library names"`
The names of one or more target libraries. If this argument isn't used, the command will pick up libraries from the `target_library` variable.

DESCRIPTION

The `analyze_minpwr_library` command examines a DC technology library from the perspective of datapath generation. Datapath generation maps to technology libraries using a pseudocell library. A pseudocell provides a specific logic function, ranging from simple gates to more complex logic cells such as full adders, all useful in generating datapath circuits.

By default, the command will list how specific pseudocell classes that are especially important to get optimal QoR map to the specified technology library. Via switches, the user can request a description of each pseudocell class.

For each pseudocell class, the output will shows whether cells in the library can support different scenarios. These scenarios include the following:

- low and high drive
- low and high area
- low and high power

Cells are checked to see if they are fully characterized, with known units, for different power usages.

EXAMPLES

The `analyze_minpwr_library` command first describes general attributes of the library.

```
Library(s) Used:
  class (File: <path>/class.db)
  Library doesn't specify units for: leakage_power
  Library units: time=1ns voltage=1V capacitance=0.100000ff dynamic_power=100nW
  Note: library doesn't contain any cells fully characterized for power.
```

This description is followed by two tables. The first table lists significant cells used in datapath generation. The four columns in the table lists whether the library contains that cell type, and the library cell that will be used in four different circumstances:

- when datapath generation is looking for a large fast cell
- when datapath generation is looking for a small cell
- when datapath generation is looking for a cell with good dynamic power

- when datapath generation is looking for a cell with good leakage power

```
=============================================================================
| For cells selected for datapath generation... |
=============================================================================
|                  | Largest          | Best Small       | Best Dynamic     | Best Leakage |
| Cell Type        | Drive            | Area Drive       | Power            | Power        |
=============================================================================
| AND2             | AN2I             | AN2I             | AN2I             | AN2I         |
| AOI21            | AO6              | AO6P             | AO6              | AO6P         |
| AOI22            | AO2              | AO2P             | AO2              | AO2P         |
| AOI222           | *NA*             | *NA*             | *NA*             | *NA*         |
| OAI21            | AO7              | AO7P             | AO7              | AO7P         |
| OAI22            | AO4              | AO4P             | AO4              | AO4P         |
| ADD_AB           | *NA*             | *NA*             | *NA*             | *NA*         |
| ADD_ABC          | *NA*             | *NA*             | *NA*             | *NA*         |
| ADD_ABCD         | *NA*             | *NA*             | *NA*             | *NA*         |
| BENC             | *NA*             | *NA*             | *NA*             | *NA*         |
| BMUX             | *NA*             | *NA*             | *NA*             | *NA*         |
| BREC             | *NA*             | *NA*             | *NA*             | *NA*         |
| BSEL             | *NA*             | *NA*             | *NA*             | *NA*         |
| BSELI            | *NA*             | *NA*             | *NA*             | *NA*         |
| BRECEN           | *NA*             | *NA*             | *NA*             | *NA*         |
| ORDEMUX          | *NA*             | *NA*             | *NA*             | *NA*         |
=============================================================================
```

The second table shows specific attributes of the cells listed in the first table.

```
=============================================================================
|                  |                  |                  | Dynanmic         | Leakage      |
| Library Cell     | Cell Area        | Cell Drive       | Power            | Power        |
=============================================================================
| AN2I             | 2.00000          | 0.12539          |      2.50000     | *INV*        |
| AO2              | 2.00000          | 0.73931          |      5.00000     | *INV*        |
| AO2P             | 4.00000          | 0.37418          |     10.00000     | *INV*        |
| AO4              | 2.00000          | 0.73931          |      5.00000     | *INV*        |
| AO4P             | 4.00000          | 0.37418          |     10.00000     | *INV*        |
| AO6              | 2.00000          | 0.73931          |      3.75000     | *INV*        |
| AO6P             | 3.00000          | 0.37418          |      7.50000     | *INV*        |
| AO7              | 2.00000          | 0.73931          |      3.75000     | *INV*        |
| AO7P             | 3.00000          | 0.37418          |      7.50000     | *INV*        |
=============================================================================
  *INV* This power value isn't characterized in the library.
```

B.3.3 report_area

Displays area information for the current design

SYNTAX: report_area [-nosplit] [-physical] [-hierarchy] [-designware]

ARGUMENTS

`-nosplit`
Prevents line-splitting and facilitates writing software to extract information from the report output. Most of the design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

`-physical`
Reports the size of the core area and the aspect ratio of the design.

`-hierarchy`
Reports the area used by cells across the design hierarchy. Reports the absolute value and the percentage of area consumed by each of the cells across the hierarchy. This option also reports the details of area contribution by combinational, non-combinational, and black box cells.

`-designware`
Reports the area of synthetic cells. The are two types of synthetic cells: The datapath cells and the DesignWare (DW) singleton cells. The datapath cells are extracted complex datapath cells. The DesignWare singleton cells are instantiated or inferred synthetic component cells. The report shows the total synthetic cell area and subtotal of datapath cell area. If the synthetic cells are ungrouped during compile, the report will show the estimated area of the ungrouped synthetic cells.

DESCRIPTION

The `report_area` command lists the current instance or current design statistics including combinational, non-combinational, and total area. If you set the `current_instance` command, the report is generated for the design of that instance. Otherwise the report is generated for the current design.

EXAMPLES

The following example generates an area report for a hierarchical design using the `-hierarchy` option:

```
prompt> report_area -hierarchy

****************************************
Report : area
Design : top
Version: Y-2006.06
Date   : Mon Dec 12 04:25:09 2005
****************************************

Library(s) Used:

    lsi_10k (File:  /usr/synopsys/libraries/tech_lib.db)

Number of ports:              13
Number of nets:               37
Number of cells:              11
Number of references:          4

Combinational area:         16.000000
Noncombinational area:     168.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:           184.000000
Total area:                 undefined

Hierarchical area distribution
------------------------------

                       Global cell area           Local cell area
                       ------------------  --------------------------
                       Absolute   Percent  combi-    noncombi-  black
Hierarchical cell      Total      Total    national  national   boxes   Design
```

```
----------------------  ---------  -------  --------  ---------  ------  ------
top                      184.0000    100.0   16.0000     0.0000  0.0000  top
mid1                      56.0000     30.4    0.0000     0.0000  0.0000  mid_0
mid1/low1                 28.0000     15.2    0.0000    28.0000  0.0000  low_0
mid1/low2                 28.0000     15.2    0.0000    28.0000  0.0000  low_5
mid2                      56.0000     30.4    0.0000     0.0000  0.0000  mid_2
mid2/low1                 28.0000     15.2    0.0000    28.0000  0.0000  low_4
mid2/low2                 28.0000     15.2    0.0000    28.0000  0.0000  low_3
mid3                      56.0000     30.4    0.0000     0.0000  0.0000  mid_1
mid3/low1                 28.0000     15.2    0.0000    28.0000  0.0000  low_2
mid3/low2                 28.0000     15.2    0.0000    28.0000  0.0000  low_1
----------------------  ---------  -------  --------  ---------  ------  ------
Total                                      16.0000   168.0000  0.0000
```

The following example generates an area report for a design which contains synthetic cells with the `-designware` option:

```
prompt> report_area -designware

****************************************
Report : area
Design : area_test
Version: D-2010.03-BETA5
Date   : Thu Jan 21 09:46:14 2010
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:
    scgp (File: /remote/testdir/libs/area_test.db)

Number of ports:            1764
Number of nets:             4643
Number of cells:            2614
Number of references:         54

Combinational area:       177312.844610
Noncombinational area:    27665.971069
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          204978.815680
Total area:                 undefined

Area of detected synthetic parts
--------------------------------
                                                    Perc. of
  Module              Implem.  Count         Area cell area
  ------------------- -------  ----- ------------ ---------
  DP_OP_2392_296_243      str      1   11454.8945      5.6%
  DP_OP_2394_297_2973     str      1   11372.6406      5.5%
  DP_OP_2396_298_8844     str      1   11372.6406      5.5%
  DP_OP_2398_299_1198     str      1    9969.5156      4.9%
  DP_OP_2400_300_4202     str      1   11183.9404      5.5%
  DP_OP_2402_301_6523     str      1    9969.5156      4.9%
  DP_OP_2404_302_2427     str      1   11372.6406      5.5%
  DP_OP_2406_303_515      str      1   11372.6406      5.5%
  DP_OP_2408_304_7752     str      1   11454.8945      5.6%
  DW01_add            apparch      1    1386.2019      0.7%
```

```
  DW02_sum            apparch      1   10547.6934      5.1%
  ------------------- -------  ----- ------------ ---------
  DP_OP Subtotal:                  9   99523.3232     48.6%
  Total:                          11  111457.2185     54.4%

Estimated area of ungrouped synthetic parts
-------------------------------------------
                            Estimated  Perc. of
  Module   Implem.  Count       Area cell area
  -------- -------  ----- ---------- ---------
  DW01_add apparch      1   541.9008      0.3%
  DW01_inc apparch      1    82.2528      0.0%
  DW01_sub apparch      9  1001.5488      0.5%
  DW_cmp   apparch      2   590.2848      0.3%
  -------- -------  ----- ---------- ---------
  Total:               13  2215.9872      1.1%

Subtotal of datapath(DP_OP) cell area:  99523.3232  48.6%  (estimated)
Total synthetic cell area:              113673.2057  55.5%  (estimated)
```

B.3.4 report_resources

Lists the resources and datapath blocks used in the design of the current instance or in the current design.

SYNTAX: report_resources [-nosplit] [-hierarchy] [-context] [-minpower]

ARGUMENTS

`-nosplit`
Prevents line-splitting and facilitates writing software to extract information from the report output. Most design information is listed in fixed-width columns. If the information for a given field exceeds the column width, the next field begins on a new line, starting in the correct column.

`-hierarchy`
Reports information about all resources used in the hierarchy of the current design or the current instance. By default, resources used only in the top level of the current design or the current instance are reported.

`-context`
Reports context information in the DesignWare generator. Currently, the power context information is printed out.

`-minpower`
Reports only DesignWare cells that are optimized for the minpower flow.

DESCRIPTION

This command lists the resources and datapath blocks used in the design of the current instance or in the current design. The command displays information about the current design or about resources and datapath blocks used in the design. If the current instance is set, the report is generated for the design of that instance. Otherwise the report is generated for the current design.

- A resource is an arithmetic or comparison operator read in as part of an HDL design. Resources can be shared when compiling the design and are reported after the compile operation completes.
- A datapath block contains one or more resources that are grouped together and optimized by a datapath generator.

- This report also has information on the parameters used to build the module, user-declared resources in each module, shared operations, and resources that are transformed into datapath blocks.
- A detailed report is printed in the DC Ultra flow. The report displays the following information about the datapath blocks in the design:
  - Design name
  - Source file location
  - Input and output connections for each datapath operand and bit width
  - Operation name with source line number
  - Datapath operation or expression

EXAMPLES

The following example displays resource and datapath information for the current design:

```
prompt> report_resources -hierarchy

****************************************
Report : resources
Design : trunc3
Version: A-2007.12-SP1
Date   : Fri Jan  4 02:33:50 2008
****************************************

Resource Report for this hierarchy in file ./designs/trunc3.v
=============================================================================
| Cell           | Module         | Parameters | Contained Operations       |
=============================================================================
| add_x_9_0      | DW01_add       | width=6    | add_9                      |
| DP_OP_6_296    | DP_OP_6_296    |            |                            |
=============================================================================

Datapath Report for DP_OP_6_296
=============================================================================
| Cell                 | Contained Operations                               |
=============================================================================
| DP_OP_6_296          | sub_8 add_8                                        |
=============================================================================
|        |      | Data     |       |                                         |
| Var    | Type | Class    | Width | Expression                              |
=============================================================================
| I1     | PI   | Unsigned | 5     |                                         |
| I2     | PI   | Unsigned | 5     |                                         |
| I3     | PI   | Unsigned | 5     |                                         |
| O1     | PO   | Signed   | 7     | I1 - I2 + I3                            |
=============================================================================

Implementation Report
=============================================================================
|                  |                  | Current          | Set              |
| Cell             | Module           | Implementation   | Implementation   |
=============================================================================
| DP_OP_6_296      | DP_OP_6_296      | str (power)      |                  |
| add_x_9_0        | DW01_add         | rpl (area,speed) |                  |
=============================================================================
```

```
No multiplexors to report

****************************************
Design : trunc3_DW01_add_0
****************************************

No resource sharing information to report.
No implementations to report
No multiplexors to report

****************************************
Design : trunc3_DP_OP_6_296_0
****************************************

No resource sharing information to report.
No implementations to report
No multiplexors to report
```

The following example displays resource and context information optimized for minpower flow:

```
prompt> report_resources -hierarchy -minpower - context

****************************************
Design : firpolyint_IN_REG1
****************************************

Minpower implementation Report
========================================================================================
|                     |                   | Current            | Set            |
| Cell                | Module            | Implementation     | Implementation |
========================================================================================
| DP_OP_24_296_7745   | DP_OP_24_296_7745 | str (power)        |                |
========================================================================================

Generator Context Report for DP_OP_24J1_296_7745
================================================================
|    Input |          |                    |               With |
|    Ports |    Width |    Variable Inputs | Switching Activity |
================================================================
|       I1 |       12 |           12  100% |            6   50% |
================================================================
```

C
DesignWare minPower FAQs

DesignWare minPower Components incorporate state-of-the-art technology to achieve low power goals. This section contains common questions asked about minPower.

C.1 minPower Licensing FAQs

This information to be supplied later.

D
Standard Synthetic Operators

Table D-5 lists the HDL operators that are mapped to synthetic operators in the Synopsys standard synthetic library `standard.sldb`. For information about the synthetic operators—input and output pins, associated modules, and so on—issue the following command:

```
dc_shell> report_synlib standard.sldb
```

Table D-5 HDL Operators Mapped to Standard Synthetic Operators

HDL Operator	Synthetic Operator(s)
+	ADD_UNS_OP, ADD_UNS_CI_OP, ADD_TC_OP, ADD_TC_CI_OP
-	SUB_UNS_OP, SUB_UNS_CI_OP, SUB_TC_OP, SUB_TC_CI_OP
*	MULT_UNS_OP, MULT_TC_OP
<	LT_UNS_OP, LT_TC_OP
>	GT_UNS_OP, GT_TC_OP
<=	LEQ_UNS_OP, LEQ_TC_OP
>=	GEQ_UNS_OP, GEQ_TC_OP
if, case	SELECT_OP

Index